МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБІГОР ЗАСОБАМИ JAVASCRIPT ТА WEBGL»**

Виконала: студентка 4 курсу, групи 6.1219-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.І. Сачковська
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, PhD, Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення

(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.

(підпис)

« 07 » 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Сачковській Аліні Ігорівні

(прізвище, ім'я та по-батькові)

1. Тема роботи (проєкту) Розробка вебігор засобами JavaScript та WebGL

керівник роботи (проєкту) Столярова Анастасія Валеріївна, PhD

(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

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

1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка вебігор засобами JavaScript та WebGL.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Кудін О.В., доцент кафедри програмної інженерії		
2	Кудін О.В., доцент кафедри програмної інженерії		
3	Кудін О.В., доцент кафедри програмної інженерії		

7. Дата видачі завдання 07.02.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.02.2023	
2.	Збір вихідних даних.	22.02.2023	
3.	Обробка методичних та теоретичних	08.03.2023	
	джерел.		
4.	Розробка першого та другого розділу.	12.04.2023	
5.	Розробка третього розділу.	16.05.2023	
6.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	21.06.2023	

Студент _____ А.І. Сачковська
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка вебігор засобами JavaScript та WebGL»: 82 с., 30 рис., 9 джерел, 2 додатки.

ВЕБГРА, ІГРОВИЙ РУШІЙ, РОЗРОБКА ІГОР, 3D-ГРАФІКА, JAVASCRIPT, WEBGL.

Об’єкт дослідження – використання WebGL для розробки вебігор.

Мета роботи: дослідження процесу розробки ігрового рушія та розробка вебгри на його основі засобами JavaScript та WebGL.

Метод дослідження – методи програмної інженерії.

У роботі проведено дослідження процесу розробки ігрового рушія з використання WebGL та JavaScript та розробка вебгри на його основі. Наведено аргументацію вибору інструментів, визначено вимоги до застосунку, побудовано відповідні діаграми та реалізовано ігровий рушій з вебгрою.

Результати роботи можуть бути використані для дослідження розробки вебігор з трьох-вимірною графікою, а також у сфері розробки ігрових рушіїв для вебзастосунків.

SUMMARY

Bachelor’s Qualifying Paper «Development of a Web Games using JavaScript and WebGL»: 82 pages, 30 figures, 9 references, 2 supplements.

WEB GAME, GAME ENGINE, GAME DEVELOPMENT, 3D-GRAPHICS, JAVASCRIPT, WEBGL.

The object of the study is using WebGL for development of a Web Games.

The aim of the study is to research process of development of game engine and development of a Web Game based on it.

The methods of research are methods of software engineering.

Work contains analysis of game engine development process using WebGL and JavaScript and development of a Web Game based on it. Given argumentation of instrument choice, defined requirements for application, built corresponding diagrams and implementation of game engine and Web Game.

Results of work can be used for research of development of Web Games with three-dimensional graphics, and also in the game engine development for Web applications sphere.

ЗМІСТ

ВСТУП

Сучасний розвиток технологій призводить до поширення інформаційних технологій в усіх сферах життя, наприклад, в сфері комп’ютерних ігор та популяризації індустрії їх розробки. В останні роки дуже популярною стає сфера веб-розробки, разом з якою зростає популярність вебігор.

Отже, актуальною задачею є дослідження та розробка ігрового рушія та гри з його використанням засобами WebGL та JavaScript.

Метою кваліфікаційної роботи є дослідження процесу розробки ігрового рушія та розробка простої вебгри на його основі засобами JavaScript та WebGL.

Задачі, які необхідно розв’язати для досягнення поставленої мети:

1) проаналізувати існуючі засоби розробки вебігор;
2) визначити необхідний функціонал ігрового рушія;
3) розробити ігровий рушій засобами WebGL та JavaScript;
4) визначити необхідний функціонал гри;
5) розробити гру, використовуючи попередньо розроблений рушій;
6) протестувати розроблений додаток.

Об’єкт дослідження – використання WebGL для розробки вебігор.

Методи дослідження – методи програмної інженерії.

Структурно дипломна робота складається з таких компонентів: вступ, 3 розділи, висновки, 2 додатки, перелік посилань з 9 найменувань.

В першому розділі проведено аналіз існуючих засобів розробки вебігор, визначено необхідний функціонал розроблюваного додатку. В другому розділі наведено процес проєктування застосунку. В третьому розділі показано процес реалізації ігрового рушія та гри з його використанням, наведено детальні пояснення механізму роботи застосунку.

1 АНАЛІЗ ВИМОГ ДО ЗАСТОСУНКУ

1.1 Аналіз засобів розробки вебігор

З активним розвитком інформаційних технологій їх популярність та затребуваність нових інструментів для розробки програмних продуктів зростає щодня. Особливої уваги потребують веб-технології, та зокрема розповсюджене відгалуження ігрової індустрії, а саме вебігри. Популярність вебігор, в порівнянні зі звичайними відео-іграми, обумовлена простим доступом до них, без необхідності завантаження, встановлення та можливих проблем із відповідністю системних вимог гри до комп’ютера, а також швидкою роботою. Вебігри може запускати та відтворювати без зависань майже будь-який сучасний браузер. Розробка вебігор є актуальною задачею, так само як і створення нових зручних інструментів для досягнення цієї мети.

Найбільш відомою та ефективною технологією для розробки вебігор є WebGL. Являючи собою низькорівневий API, що базується на популярному стандарті розробки мобільних додатків з трьох вимірною (далі 3D) графікою OpenGL ES 2.0, WebGL дозволяє створювати та використовувати 3D графіку з елементом HTML «canvas» у всіх браузерах, що підтримують цю технологію [1, с. 8]. WebGL використовує мову шейдерів, що також є елементом HTML, для відтворення 3D графіки у браузері, та може працювати із мовою програмування JavaScript. Однак, WebGL не є повноцінним ігровим рушієм, а лише інструментом, на основі якого можна створити власний ігровий рушій, із задоволенням необхідних технічних потреб та будь-яким ступенем індивідуалізації.

На основі WebGL вже існують більш високо рівневі бібліотеки, такі як Three.js, що використовують функції вище зазначеної технології для створення сцени, 3D об’єктів, камери, освітлення, тощо. Також існує певна кількість готових ігрових рушіїв створених з використанням WebGL, такі як

Unreal Engine або Cocos2D, що дозволяють розробнику швидко створювати власний продукт, проте вони не є повноцінними застосунками для розробки саме вебігор.

Також альтернативою WebGL є Adobe Flash, система для відображення мультимедійних матеріалів у браузері, включаючи веб-графіку. Однак, не зважаючи на свою популярність у минулому, цей продукт має не такий великий обсяг можливостей, є застарілим та більше не підтримується у більшості браузерів.

Таким чином чітко зрозуміло, що WebGL є популярним засобом, призначеним для створення та відтворення 3D графіки в браузері, є швидким та підтримується майже в усіх браузерах. Вище зазначений API ідеально підходить для розробки ігрового рушія для створення вебігор з сучасною графікою, тому актуальною задачею є дослідження процесу розробки ігрового рушія на основі WebGL, для повноцінного розуміння роботи програми та можливості індивідуального налаштування рушія під необхідні вимоги.

Однак, для реалізації повноцінного ігрового рушія, завантаження та відтворення графіки не достатньо. Для функціонування гри необхідна фізика, що дозволить ігровим об’єктам реалістично взаємодіяти один з одним та з оточуючим середовищем, тому наступним кроком є вибір підходящої JavaScript бібліотеки, що дозволяє створювати ігрову фізику. Щоб обрати найбільш підходящу бібліотеку потрібно провести порівняльний аналіз існуючих фізичних бібліотек.

Ammo.js це фізична бібліотека, що є повною копією іншої бібліотеки Bullet написаної на C++, як зазначає джерело [1]. Ця бібліотека була автоматично переведена на мову програмування JavaScript, так само як і документація до неї, тому її використання може викликати складнощі. Окрім того, Ammo.js не була розроблена для використання з JavaScript, тому ця бібліотека може знизити швидкість роботи програми. Хоча Ammo.js містить величезну кількість фізичних функцій, вона не є оптимальним варіантом для використання.

JigLibJS також є портом фізичної бібліотеки JigLib з мови C++, проте ця бібліотека вручну переписана та перероблена для роботи на JavaScript [1]. JigLibJS має високі показники швидкості роботи, проте бракує певної кількості функцій, на відміну від попередньої бібліотеки, а також не має належної документації та є застарілою, оскільки не підтримується розробником протягом довгого часу.

Cannon.js це фізична бібліотека написана на JavaScript, що має багато необхідних функцій, оскільки заснована на Ammo.js. Бібліотека розроблена спеціально для JavaScript та веб-розробки, тому не сповільнює роботу програми, а також є зручною для роботи з WebGL. Також Cannon.js є сучасною бібліотекою, що підтримується автором, а також має зручну документацію та багато інших матеріалів для полегшення роботи з бібліотекою.

Таким чином на основі проведеного порівняння деяких існуючих фізичних бібліотек, можна обрати найбільш підходящу для досягнення поставлених цілей. В даному випадку найкращим варіантом є Cannon.js з її функціональністю, швидкістю та зручністю використання.

Отже, зважаючи на всю вище зазначену інформацію, для розробки власного ігрового рушія та гри на його основі буде використовуватися API WebGL для створення та відображення 3D графіки та бібліотека Cannon.js для імплементації фізики у гру.

1.2 Визначення функціональних вимог

Перед початком розробки будь-якого програмного застосунку в першу чергу необхідно визначити вимоги до майбутнього продукту, а саме які функції він повинен виконувати та які можливості надавати користувачам.

Оскільки метою роботи є, перш за все, розробка ігрового рушія з використанням WebGL та Cannon.js, що дозволить користувачам без перешкод створювати власні вебігри використовуючи наявний функціонал, то

розроблюваний додаток повинен мати такі основні можливості:

– створення ігрової сцени, до якої можна буде додавати об’єкти;

– завантаження власних 3D моделей;

– редагування позиції, куту обертання та масштабування об’єкту;

– завантаження текстур моделей;

– створення джерел освітлення, зміна їх позиції, напрямку та кольору;

– створення камери, встановлення її позиції та напрямку відображення ігрового світу;

– створення фізики у грі, встановлення гравітації;

– присвоєння 3D моделям фізичних тіл, завдяки яким об’єкти будуть взаємодіяти один з одним;

– надання кожному фізичному тілу маси, форми та розміру;

– оновлення фізичного світу та графічного зображення через певні проміжки часу для синхронізації змін із їх відображенням;

– оновлення кожної моделі у ігровому світі для відповідності стану моделі її фізичному тілу.

Таким чином з цього переліку чітко зрозуміло, які саме можливості користувача та функції необхідно реалізувати у майбутньому продукті.

Окрім ігрового рушія, метою роботи є також розробка простої гри з його використанням, для демонстрації можливостей та наявних функцій створеної програми. Тому наступним кроком є визначення необхідного функціоналу майбутньої вебгри.

Ігровий процес розроблюваного додатку представляє собою симулятор кухара в кафе з десертами: кухарю надходять замовлення від клієнтів, а він в свою чергу повинен правильно зібрати замовлення та віддати відвідувачу. Ігровий процес не є занадто складним, оскільки метою роботи є демонстрація можливостей розробленого ігрового рушія.

Отже, для створюваної вебгри необхідно забезпечити такі функції та

можливості користувача:

– можливість керування кухаром за допомогою клавіш, його переміщення та обертання в необхідному напрямку;

– можливість брати страви, представлені на кухні, або піднос для видачі замовлень, переміщувати їх разом із персонажем;

– викидання предметів на підлогу або розміщення їх на столі або підносі;

– функція викидання зайвих предметів або неправильного замовлення за допомогою смітника;

– надходження та відображення випадкових замовлень на стендах;

– при видачі замовлення функція перевірки правильності замовлення – якщо воно вірне, то з’являється нове замовлення, в іншому випадку замовлення необхідно робити з початку.

Зазначені вище функції є необхідними для функціонування гри та демонстрації можливостей створюваного ігрового рушія.

Таким чином було визначено та описано функціональні вимоги до майбутнього програмного продукту – ігрового рушія, заснованого на WebGL та Cannon.js, а також простою гри, розробленої за допомогою створеної програми. Процес реалізації всіх зазначених раніше функцій застосунку описано у наступних розділах.

1.3 Висновки до розділу 1

В розділі 1 було наведено аналіз існуючих технологій для розробки вебігор, обрано та описано засоби (WebGL та Cannon.js), що будуть використані для досягнення поставлених у роботі цілей. Також було визначено та описано весь необхідний функціонал та можливості користувача у розроблюваному програмному застосунку.

2 ПРОЄКТУВАННЯ ЗАСТОСУНКУ

2.1 Проєктування структури застосунку

Після вибору технологій для реалізації застосунку, а також визначення функціональних вимог та можливостей користувача, наступним етапом є проєктування розроблюваного додатку.

Для демонстрації структури майбутнього ігрового рушія буде використано UML діаграму класів, розроблену за рекомендаціями джерела [3]. На рисунку 2.1 зображено діаграму класів ігрового рушія, що показує необхідні класи для задоволення всіх зазначених раніше вимог до програми та можливостей для подальшої реалізації вебгри.

Оскільки метою роботи є не лише розробка ігрового рушія, а також простої вебгри з його використанням, то наступним кроком є розробка UML діаграми класів для ігрових класів, що реалізуватимуть ігрові об’єкти та їх логіку. Слід зауважити, що всі класи, продемонстровані на діаграмі, є нащадками класу StageObject, зображеного на рис. 2.1, а тому мають ті самі властивості та функції що й батьківський клас. Отримана діаграма зображена на рисунку 2.2.

Окрім діаграми класів розроблювана вебгра потребує структурованої демонстрації процесу гри та можливостей гравця. Тому необхідно створити UML діаграму активності, як показано у [3], щоб чітко розуміти процес та майбутню логіку гри та як саме будуть задоволені вимоги, визначені у розділі 1.

Діаграма активності гри демонструє, як саме гравець буде взаємодіяти з ігровим оточенням, що він може робити та що буде відбуватися, як буде змінюватися стан об’єктів у ігровому світі у відповідь на певні дії та ситуації (див. рис. 2.3).

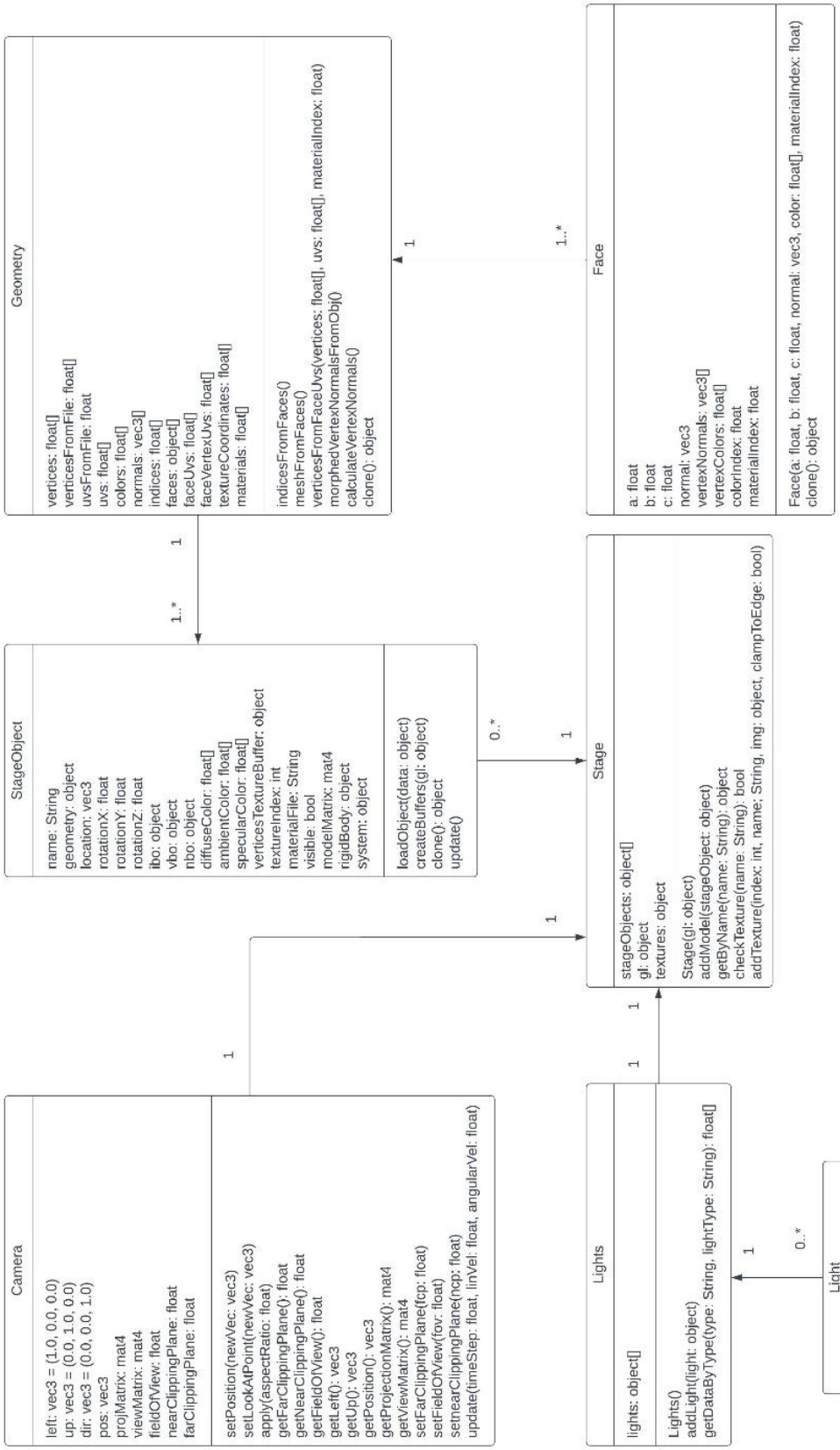

Рисунок 2.1 – UML діаграма класів ігрового рушія

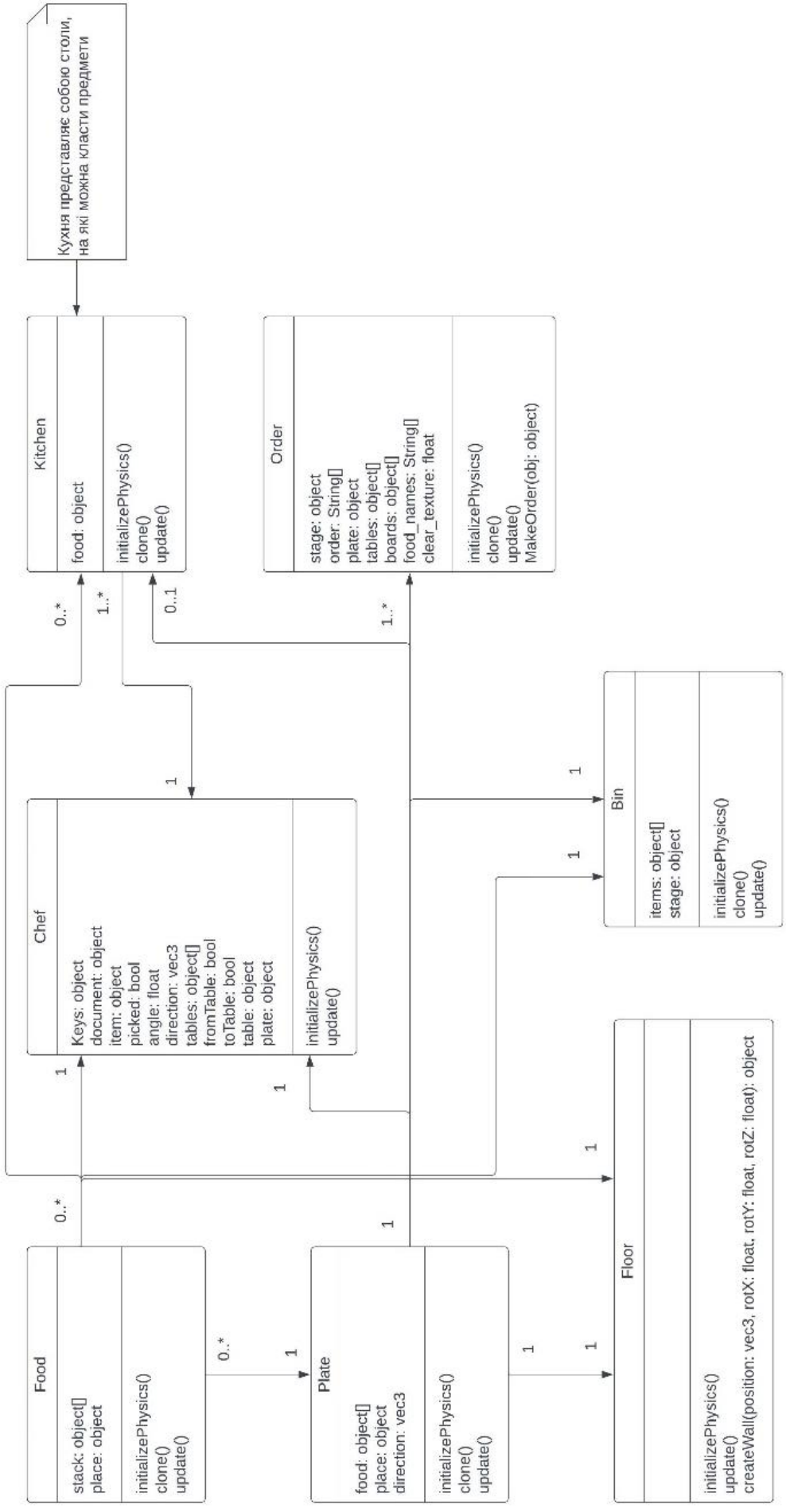

Рисунок 2.2 – UML діаграма класів ігрової логіки

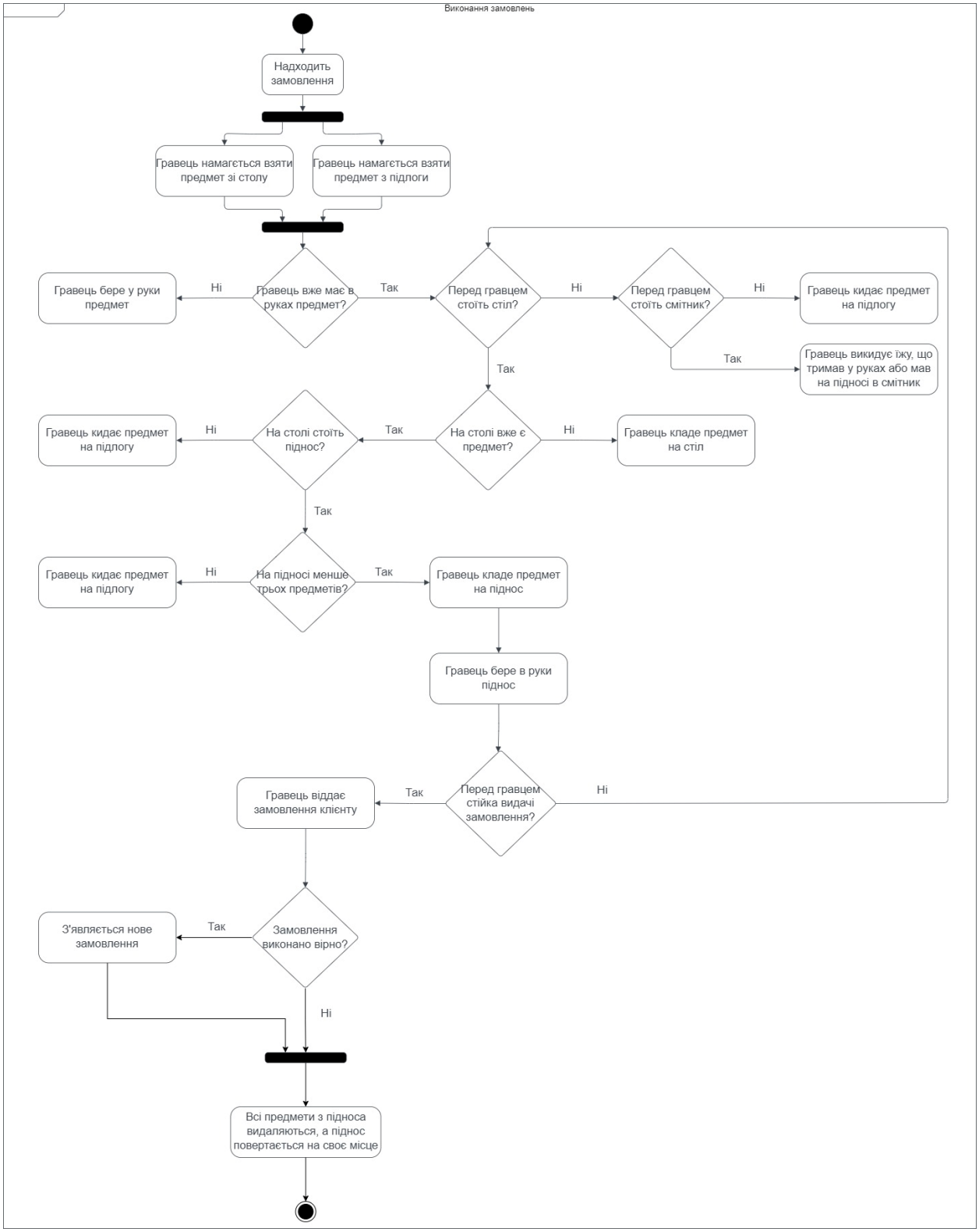

Рисунок 2.3 – UML діаграма активності ігрового процесу

Таким чином було спроєктовано майбутній ігровий рушій та вебгру на його основі, розроблено необхідні UML діаграми для демонстрації того, як будуть задовільнені вимоги до застосунку.

2.2 Висновки до розділу 2

У розділі 2 було спроєктовано майбутній застосунок та побудовано відповідні UML діаграми для демонстрації структури проєкту. Було розроблено діаграму класів ігрового рушія, так само як і для ігрової логіки на його основі. Окрім того, було побудовано діаграму активності для ігрової логіки, щоб показати як саме буде працювати майбутня гра та як користувач може взаємодіяти з ігровим світом. Усі наведені діаграми демонструють, як у програмі буде реалізовано визначені у попередньому розділі вимоги до застосунку та можливості користувача.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Програмна реалізація ігрового рушія

Після вибору інструментів для розробки застосунку, аналізу функціональних вимог та можливостей користувача, а також проєктування майбутньої програми, наступним кроком є програмна реалізація.

WebGL працює з HTML-елементом «canvas», отже в першу чергу необхідно створити HTML-документ з відповідним елементом в ньому. Щоб підключити до нього WebGL, потрібно створити «script» для роботи з JavaScript, отримати елемент «canvas» за допомогою його ідентифікатора та передати до функції ініціалізації WebGL [1]. У цій функції, використовуючи різні варіанти назви WebGL API, програма намагається визначити контекст елементу «canvas» (рис.3.1). Якщо WebGL не підтримується у браузері, виводиться відповідне повідомлення.

```
function initGL(canvas) {

    var names = ["webgl", "experimental-webgl", "webkit-3d", "moz-webgl"];

    for (var i = 0; i < names.length; ++i) {
       try {
          gl = canvas.getContext(names[i]);
       }
       catch (e) { }
       if (gl) {
          break;
       }
    }
    if (gl == null) {
       alert("Could not initialise WebGL");
       return null;
    }

    gl.viewportWidth = canvas.width;
    gl.viewportHeight = canvas.height;
}
```

Рисунок 3.1 – Ініціалізація WebGL

Графіка у WebGL створюється за допомогою шейдерів – це програми, що передають дані у графічний процесор комп’ютера, завдяки чому на ньому відтворюється графіка. WebGL використовує два типи шейдерів: шейдер вершин та шейдер фрагментів.

Перший працює з даними вершин, такими як позиція, нормалі, кольори і т. д. Другий в свою чергу працює з даними фрагментів (пікселів), та використовуючи дані з шейдеру вершин та власні дані визначає колір кожного пікселя на екрані.

Для роботи з шейдерами необхідно використовувати мову шейдерів GLSL.

На рисунку 3.2 та 3.3 показано реалізацію обох шейдерних програм [1], що з використанням необхідних даних розраховують позиції вершин та колір кожного пікселя на екрані.

```
        <script id="shader-vs" type="x-shader/x-vertex">
           attribute vec3 aVertexPosition;
           attribute vec3 aVertexNormal;
           uniform mat4 mVMatrix;
           uniform mat4 pMatrix;
           uniform mat4 nMatrix;
           varying vec3 transformedNormal;
           varying vec3 vertexPos;
           attribute vec2 aTextureCoord;
           varying highp vec2 vTextureCoord;
           uniform bool hasTexture;
           void main(void) {
           vec4 vertexPos4 =  mVMatrix * vec4(aVertexPosition, 1.0);
           vertexPos = vertexPos4.xyz;
           transformedNormal = vec3(nMatrix * vec4(aVertexNormal,1.0));
           gl_Position= pMatrix *vertexPos4;
            if(hasTexture){
             vTextureCoord = aTextureCoord;
            }
           }
</script>
```

Рисунок 3.2 – Шейдер вершин

```
<script id="shader-fs" type="x-shader/x-fragment">
   precision mediump float;
   varying vec3 transformedNormal;
   varying vec3 vertexPos;
   uniform vec3 uAmbientColor;
   uniform vec3 uLightingPosition;
   uniform vec3 uDirectionalColor;
   uniform vec3 uSpecularColor;
   uniform vec3 materialDiffuseColor;
   uniform vec3 materialAmbientColor;
   uniform vec3 materialSpecularColor;
   uniform sampler2D uSampler;
   varying highp vec2 vTextureCoord;
   uniform bool hasTexture;
   void main(void)  {
   vec3 normal=normalize(transformedNormal);
   vec3 eyeVector=normalize(-vertexPos);
   vec3 lightDirection = normalize(uLightingPosition);
   float specular = 0.0;

   float directionalLightWeighting = max(dot(normal, -lightDirection), 0.0);
   if(directionalLightWeighting>0.0)
   {
   vec3 halfDir = normalize(-lightDirection + eyeVector);
   float specAngle = max(dot(halfDir, normal), 0.0);
   specular = pow(specAngle, 4.0);
   }
   if(hasTexture){
   vec3 iColor = uAmbientColor+ uDirectionalColor*
directionalLightWeighting+uSpecularColor*materialSpecularColor*specular;
   vec4 tColor=texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
   gl_FragColor = vec4(iColor, 1.0)*texture2D(uSampler, vec2(vTextureCoord.s,
vTextureCoord.t));
   }
   else{
   vec3 iColor = uAmbientColor*materialAmbientColor+ uDirectionalColor
*materialDiffuseColor *
directionalLightWeighting+uSpecularColor*materialSpecularColor*specular;
   gl_FragColor = vec4(iColor, 1.0);
   }
   }
</script>
```

Рисунок 3.3 – Шейдер фрагментів

Щоб шейдери могли працювати та отримувати інформацію, потрібно зробити їх ініціалізацію. Шейдери підключаються до однієї спільної програми

та пов’язуються для подальшої взаємодії. Також створюється посилання на атрибути шейдерів, щоб пізніше їх можна було зв’язати з буферами, що міститимуть дані, як це показано у [1]. Частина функції ініціалізації продемонстрована на рисунку 3.4.

```
            var fragmentShader = getShader(gl, "shader-fs");
            var vertexShader = getShader(gl, "shader-vs");

            shaderProgram = gl.createProgram();
            gl.attachShader(shaderProgram, vertexShader);
            gl.attachShader(shaderProgram, fragmentShader);
            gl.linkProgram(shaderProgram);

            if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
               alert("Could not initialise shaders");
            }

            gl.useProgram(shaderProgram);

            shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram,
"aVertexPosition");
            gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
            shaderProgram.vertexNormalAttribute = gl.getAttribLocation(shaderProgram,
"aVertexNormal");
            gl.enableVertexAttribArray(shaderProgram.vertexNormalAttribute);

            shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram,
"pMatrix");
            shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram,
"mVMatrix");
            shaderProgram.nMatrixUniform = gl.getUniformLocation(shaderProgram,
"nMatrix");
            shaderProgram.ambientColorUniform = gl.getUniformLocation(shaderProgram,
"uAmbientColor");
            shaderProgram.specularColorUniform = gl.getUniformLocation(shaderProgram,
"uSpecularColor");
```

Рисунок 3.4 – Ініціалізація шейдерів

Щоб шейдери могли отримувати інформацію для кожного об’єкта створюється буфери, що зберігатимуть дані, такі як вершини, індекси вершин, нормалі. На рисунку 3.5 показано приклади створення буферів вершин та індексів для об’єкта, а також внесення даних у них [1].

```
        this.vbo = gl.createBuffer();
            this.ibo = gl.createBuffer();

            gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, this.ibo);
            gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new
Uint16Array(this.geometry.indices), gl.STATIC_DRAW);
            gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, null);

            gl.bindBuffer(gl.ARRAY_BUFFER, this.vbo);
            gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(this.geometry.vertices),
gl.STATIC_DRAW);
            this.vbo.itemSize = 3;
            this.vbo.numItems = this.geometry.vertices.length/3;
```

Рисунок 3.5 – Створення буферів

Оскільки створення ігор без використання текстур є майже неможливим, їх завантаження є наступним кроком у реалізації рушія. Для завантаження текстур також використовуються буфери, що зберігають їх дані (рис. 3.6) [1].

```
        this.gl.bindTexture(this.gl.TEXTURE_2D, texture.texture);

            this.gl.texImage2D(this.gl.TEXTURE_2D, 0, this.gl.RGBA, this.gl.RGBA,
this.gl.UNSIGNED_BYTE, img);
            this.gl.texParameteri(this.gl.TEXTURE_2D, this.gl.TEXTURE_MAG_FILTER,
this.gl.NEAREST);
            this.gl.texParameteri(this.gl.TEXTURE_2D, this.gl.TEXTURE_MIN_FILTER,
this.gl.NEAREST);
            if(clampToEdge){
               this.gl.texParameteri(this.gl.TEXTURE_2D, this.gl.TEXTURE_WRAP_S,
this.gl.CLAMP_TO_EDGE);
               this.gl.texParameteri(this.gl.TEXTURE_2D, this.gl.TEXTURE_WRAP_T,
this.gl.CLAMP_TO_EDGE);

            }
            this.gl.bindTexture(this.gl.TEXTURE_2D, null);
```

Рисунок 3.6 – Створення буферу для завантаження текстури

Поєднанням текстури з об’єктом займаються шейдери. Із даних об’єкта вершинний шейдер дізнається, які координати текстури відповідають певній вершині, а шейдер фрагментів розраховує, якому кольору на текстурі буде відповідати кожен піксель між двома вершинами. На рисунку 3.6 показано встановлення параметрів вибору кольору на текстурі шейдером фрагментів.

Щоб керувати відображенням об’єктів у ігровому світі використовується клас Camera. Він розраховує та зберігає матрицю вигляду (відповідає за трансформацію ігрових об’єктів у світі відносно камери, наприклад обертання об’єкту при повороті чи переміщенні камери), а також матрицю проекції (відповідає за масштабування об’єктів, розраховується із параметрів камери, таких як найближча відстань яку бачить камера та найвіддаленіша, а також кут огляду камери). При розрахуванні відображення об’єктів використовуються саме ці дві матриці. Для роботи з матрицями використовується бібліотека gl-matrix-min.js. Також клас камери містить функції для задання позиції камери, направлення камери (задається точкою у світі), та інші функції для задання та отримання параметрів камери, які показано на діаграмі класів (рис. 2.1). На рисунку 3.7 показано функцію розрахунку матриці вигляду та матриці проекції, за інструкцією джерела [1].

```
        var matView=mat4.create();
        var lookAtPosition=vec3.create();
        vec3.add(lookAtPosition,this.pos, this.dir);
        mat4.lookAt(matView, this.pos, lookAtPosition, this.up);
        mat4.translate(matView,matView,vec3.fromValues(-this.pos[0], -this.pos[1], -
this.pos[2]));
        this.viewMatrix = matView;

        this.projMatrix=mat4.create();
        mat4.perspective(this.projMatrix, degToRadian(this.fieldOfView), aspectRatio,
this.nearClippingPlane, this.farClippingPlane);
```

Рисунок 3.7 – Створення матриць відображення

Після всіх попередніх кроків можна переходити до завантаження об’єктів на ігрову сцену. 3D-моделі складаються з полігонів, в більшості випадків – трикутників, тому розроблюваний ігровий рушій буде працювати саме з ними. Кожен полігон містить такі параметри, як вершини, їх позиції, кольори, координати текстур, нормаль полігону, нормалі вершин та інших. Оскільки вручну створювати моделі шляхом запису координат їх вершин задача занадто складна та не практична, моделі розробляють у спеціальних

програмах. Тому для того, щоб отримати дані про модель необхідно перетворити файл з об’єктом у зручний для обробки формат, а саме JSON. Для цього використовується спеціальний скрипт convert_obj_three.py, що знаходиться у вільному доступі.

Далі для зберігання даних кожної моделі необхідно розробити класи Face та Geometry, що представляють полігон та геометрію моделі відповідно. Для завантаження даних у класи потрібно створити функції, що розбирають JSON-файл та записують інформацію у ці класи, аналогічно до показаного у [1]. Реалізація оговорених класів та функцій наведена у додатку А.

Окрім того, для збереження самого об’єкта створюється клас StageObject, що зберігає всі дані про об’єкт на сцені, що реалізується за допомогою класу Stage та містить масив об’єктів класу StageObject. На рисунку 3.8 наведено функцію завантаження об’єкта, а також розбір даних про його матеріал (кольори та текстура).

```
StageObject.prototype.loadObject= function (data){
    this.geometry=parseJSON(data);
    this.name=data.metadata.sourceFile.split(".")[0];

    if(this.geometry.materials.length>0){
        try {
            if(!(this.geometry.materials[0].colorDiffuse===undefined))
                this.diffuseColor=this.geometry.materials[0].colorDiffuse;

        } catch (error) {
            console.log(error);
        }
        if(!(this.geometry.materials[0].colorAmbient===undefined))
            this.ambientColor=this.geometry.materials[0].colorAmbient;
        if(!(this.geometry.materials[0].mapDiffuse===undefined)){
            this.materialFile=this.geometry.materials[0].mapDiffuse;
        }
    }
}
```

Рисунок 3.8 – Завантаження об’єкта

Також необхідним елементом на сцені є освітлення, оскільки без нього жоден об’єкт не буде відображатися. Для додавання освітлення створено клас

Light, та клас для зберігання кількох джерел освітлення Lights. Нижче показано, як встановлюються параметри джерела світла у шейдери (рис. 3.9) [1], що вже містять код для їх обробки.

```
gl.uniform3f(shaderProgram.ambientColorUniform,0.5,0.5,0.55);
gl.uniform3f(shaderProgram.specularColorUniform,1.0,1.0,1.0);
var lightingDirection = [0,-1.25,-1.25];
gl.uniform3fv(shaderProgram.lightingDirectionUniform, lightingDirection);
gl.uniform3f(shaderProgram.directionalColorUniform,0.5,0.5,0.55);
```

Рисунок 3.9 – Створення джерела світла

Після того, як всі необхідні класи та функції підготовлено, можна завантажити бажані об’єкти та подивитися, як буде виглядати майбутня вебгра (рис. 3.10).

Коли сцена створена, всі моделі завантажено та розміщено, необхідно реалізувати процес оновлення ігрової сцени через певний проміжок часу, оскільки пізніше в процесі гри стан об’єктів буде постійно змінюватися і його необхідно своєчасно відображати. Для цього було розроблено функцію анімації, що перевіряє скільки часу пройшло, та якщо він більший за зазначений час оновлення – викликає функцію для оновлення стану всіх об’єктів на сцені (див. рис. 3.11) [1].

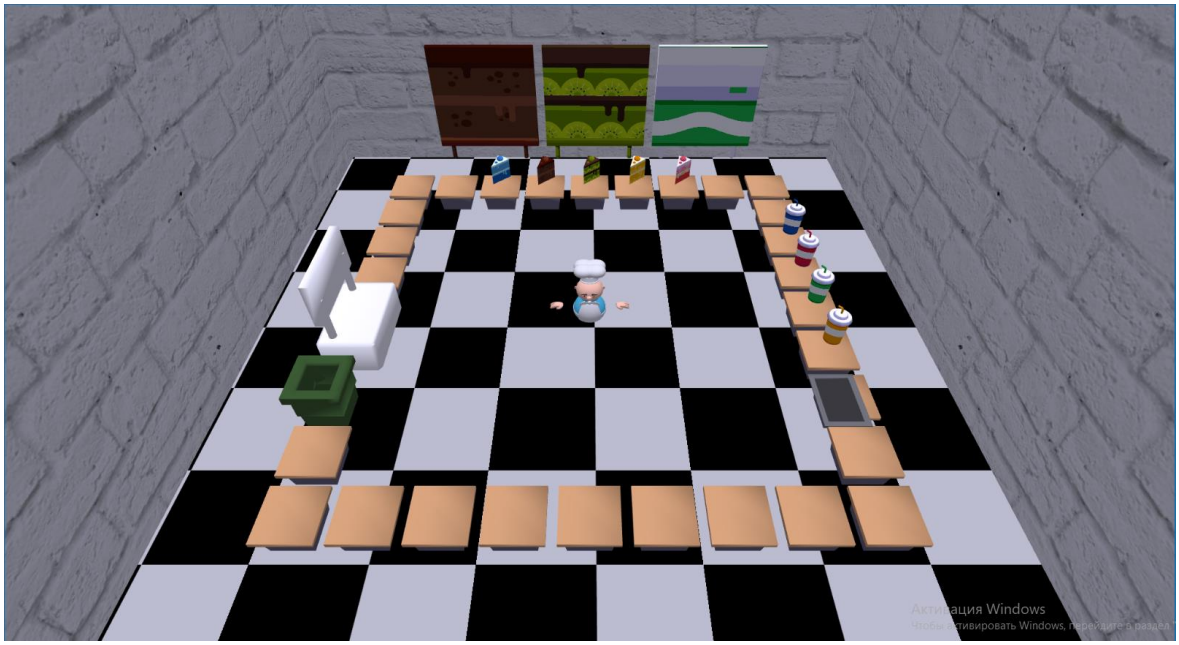

Рисунок 3.10 – Завантажені об’єкти

```
function animate(){
currentTime = (new Date).getTime();
elapsedTime = currentTime - lastFrameTime;
if (elapsedTime < rate) return;
var updateIterations = Math.floor(elapsedTime / MINIMUM_FRAME_RATE);
while(updateIterations > 0){
try{
drawScene();
}
catch(e){
console.log(e);
}
updateIterations -= 1;
}
lastFrameTime = (new Date).getTime();
}
```

Рисунок 3.11 – Функція анімації

Також клас StageObject має функцію оновлення, що викликається у попередній функції. В ній відбувається зміна графічного відображення моделі у відповідності до матриці вигляду світу та власних параметрів об’єкта, таких як позиція та обертання, за джерелом [1]. В результаті цих обчислень змінюється власна матриця вигляду об’єкту (рис. 3.12).

```
StageObject.prototype.update=function(){
    mat4.identity(this.modelMatrix);

    mat4.translate(this.modelMatrix,this.modelMatrix, this.location);

    mat4.rotateX(this.modelMatrix,this.modelMatrix,this.rotationX);
    mat4.rotateY(this.modelMatrix,this.modelMatrix,this.rotationY);
    mat4.rotateZ(this.modelMatrix,this.modelMatrix,this.rotationZ);

  }
```

Рисунок 3.12 – Функція оновлення об’єкта

Таким чином реалізовано відображення 3D-графіки у HTML-елементі «canvas», можливість завантажувати моделі, текстури, задавати власні параметри моделей, додавання освітлення та встановлення параметрів камери, а також оновлення ігрового світу. Імплементація фізики у гру буде розглянута у підрозділі 3.2, разом із реалізацією ігрової логіки та всіма необхідними функціями вебгри.

3.2 Програмна реалізація вебгри

Після реалізації графічної частини ігрового рушія, можна переходити до розробки вебгри на його основі. Разом зі створенням ігрових об’єктів буде додано фізичні моделі до них та взаємодію фізичних об’єктів один з одним.

Щоб працювати з функціями Cannon.js в першу чергу необхідно створити фізичний світ. Для цього створюється змінна світу, до якого додається гравітація в необхідному напрямку, у даному випадку це від’ємна вісь Y (рис. 3.13) [4].

```
function initCannon() {
    system = new CANNON.World();
    system.gravity.set(0,  -9.8, 0); // m/s²
  }
```

Рисунок 3.13 – Ініціалізація фізичного світу

Коли фізичний світ створено, наступним кроком є розробка класів для ігрових об’єктів. Перш за все необхідно додати підлогу та стіни, які будуть обмежувати переміщення предметів у грі. Для цього додається клас Floor, що містить у собі 3D-модель підлоги. Щоб підлога не дозволяла іншим об’єктам провалюватись крізь нього, потрібно додати йому фізичне тіло за допомогою Cannon.js, як показано на рисунку 3.14. Для тіла встановлюється форма (куб, сфера, поверхня, тощо), розмір, маса, позиція та обертання [4]. Те саме повторюється і для стін.

Також за тим самим принципом створюється клас Kitchen, екземпляром якого є кухонний стіл, на який можна ставити предмети. Різниця з попереднім прикладом фізичного тіла є те, що для стола формою тіла є куб, а не площина. Також клас Kitchen має змінну для зберігання предмета, який стоїть на столі.

Аналогічно створюється і клас Food, що містить екземпляри страв, що подаватиме кухар. Цей клас має змінну для зберігання стеку копій об’єктів цього класу, для тих страв, що з самого початку розміщені на кухні. Коли

гравець бере предмет з цього стеку, головний об’єкт залишається на місці, а гравець отримує лише його копію. Таким чином імітується велика кількість однакових страв.

```
var groundBody = new CANNON.Body({
    mass: 0,
    position: new CANNON.Vec3(0, 0, 0),
    collisionFilterGroup: GROUP2,
    collisionFilterMask: GROUP2
});
var groundShape = new CANNON.Plane();
groundBody.addShape(groundShape);
groundBody.quaternion.setFromEuler(-Math.PI / 2, 0, 0);
this.rigidBody=groundBody;
this.system.addBody(groundBody);
```

Рисунок 3.14 – Додавання фізичного тіла до об’єкта

На рисунку 3.15 наведено функцію клонування страв та зберігання їх у стек.

```
if(food_names.includes(stageObject.name)){
        var items=[];
        for(var j=0;j<3;++j){
            var item=stageObject.clone();
            items.push(item);
            item.system=system;
            item.visible = false;

            stage.addModel(item);
        }
        stageObject.stack = items;
    }
```

Рисунок 3.15 – Клонування страв

Так само, як і попередні об’єкти, розроблюється клас Plate, екземпляром якого є піднос, на який необхідно класти страви перед видачою замовлення клієнту. Цей клас має змінну-масив для зберігання до трьох екземплярів класу

Food, що імітує відповідно процес складання елементів замовлення на піднос та зберігання їх на ньому. Також піднос містить вектор орієнтації, що зберігає вісь обертання підносу.

Наступним кроком є розробка одного з найважливіших класів, а саме класу Chef, екземпляром якого є об’єкт кухара – модель якою буде керувати гравець. У функції update() цього класу встановлено слухач подій, що зберігає у відповідному об’єкті стан натискання клавіш управління персонажем, а саме WASD. Відповідно до натиснутих клавіш код на рисунку 3.16 змінює позицію кухара, а також записує до вектору направлення об’єкту – наприклад, якщо натиснута клавіша A, персонаж переміщується вліво та повинен повернутися у той самий бік.

```
var pos = this.rigidBody.position;
if(this.Keys.up){
   this.rigidBody.position.z-=1;
   this.direction.z=-1;
   if(!this.Keys.left&&!this.Keys.right) this.direction.x=0;
}
if(this.Keys.down){
   this.rigidBody.position.z+=1;
   this.direction.z=1;
   if(!this.Keys.left&&!this.Keys.right) this.direction.x=0;
}
if(this.Keys.left) {
   this.rigidBody.position.x-=1;
   this.direction.x=-1;
   if(!this.Keys.up&&!this.Keys.down) this.direction.z=0;
}
if(this.Keys.right){
   this.rigidBody.position.x+=1;
   this.direction.x=1;
   if(!this.Keys.up&&!this.Keys.down) this.direction.z=0;
```

Рисунок 3.16 – Переміщення персонажа

Також на рисунку 3.16 показано, як розраховується бажаний кут оберту моделі, відповідно до натиснутих клавіш та попереднього значення обертання об’єкту.

```
var ax, az, tx, tz;
ax = this.location[0];
az = this.location[2];
tx = this.rigidBody.position.x;
tz = this.rigidBody.position.z;

const desiredAngle = Math.atan2(tx-ax, tz-az);
if(Object.values(this.Keys).includes(true)){
   this.rotationY=desiredAngle;
   this.angle = this.rotationY;
}
mat4.rotateY(this.modelMatrix,this.modelMatrix,this.rotationY);
this.rigidBody.quaternion.setFromEuler(0, this.angle, 0);
```

Рисунок 3.17 – Обертання персонажа

Коли персонаж може рухатися, необхідно реалізувати можливість користувача брати та класти предмети. Щоб підібрати предмет, необхідно визначити, який саме предмет буде використано. Для цього у бібліотеці Cannon.js існує клас Ray, що представляє собою промінь, якому можна задати вихідну та кінцеву точку. Коли промінь перетинає ігровий фізичний світ заповнюється його властивість RaycastResult, що відповідно містить у собі результат перетину світу променем, включаючи найближче фізичне тіло, з яким зіткнувся промінь. Створення променю, що виходить із точки місцезнаходження ігрового персонажа, та йде в напрямку його повертання, показано на рисунку 3.18 [4].

```
const from = new CANNON.Vec3(this.rigidBody.position.x+this.direction.x*2, 1, this.rigidBody.position.z+this.direction.z*2);
const to = new CANNON.Vec3(this.rigidBody.position.x+this.direction.x*5, 1, this.rigidBody.position.z+this.direction.z*5);
var ray = new CANNON.Ray(from, to);
ray.intersectWorld(this.system, {mode: CANNON.Ray.CLOSEST, collisionFilterGroup: GROUP3, collisionFilterMask: GROUP2 | GROUP4});
```

Рисунок 3.18 – Створення променю

За результатом RaycastResult програма перебирає різні варіанти подій, описані на діаграмі активності (див. рис. 2.3). Якщо промінь стикається зі столом, на якому є предмет, користувач може взяти цей предмет. При

натисканні клавіші E перевіряється, чи перетинає промінь такий стіл у даний момент – якщо так, то предмет зі столу закріпляється на невеликій відстані перед кухаром (також враховується напрямок, у який розвернутий гравець), імітуючи те, що він тримає предмет перед собою (рис. 3.19). Якщо предмет – це страва, що має стек власних копій для використання, то при взаємодії в цим предметом одна копія береться гравцем та видаляється зі стеку.

```
        if(ray.hasHit&&!this.picked){
            for(var t in this.tables){
              if(tables[t].rigidBody==ray.result.body && tables[t].food && tables[t].food.stack)
{
                this.fromTable = true;
                this.item = tables[t].food.stack;
                this.table = null;
                break;
              }
            }
          }

        if(chef.fromTable&&chef.table==null){
                chef.item = chef.item.pop();
                                                chef.item.place = vec3.fromValues(
chef.rigidBody.position.x+chef.direction.x*3,                   chef.rigidBody.position.y,
chef.rigidBody.position.z+chef.direction.z*3);
                                              chef.item.location = vec3.fromValues(
chef.rigidBody.position.x+chef.direction.x*3,                   chef.rigidBody.position.y,
chef.rigidBody.position.z+chef.direction.z*3);
                chef.item.initializePhysics();
                chef.item.visible = true;
                chef.item = chef.item.rigidBody;
                chef.fromTable = false;
              }
```

Рисунок 3.19 – Підбирання предмету зі столу

Аналогічним чином проводиться перевірка всіх інших варіантів взаємодії гравця з об’єктами. Наприклад, якщо стіл, який перетинає промінь, пустий – нічого не відбудеться, а якщо на столі стоїть піднос, створення якого було описано раніше, то копії не створюється, а предмет просто закріпляється перед кухаром.

Коли предмети можна брати, то необхідно мати можливість їх класти.

Так само як і з підбиранням предметів – промінь перетинає світ, та можна дізнатися, з яким саме об’єктом він стикається. Якщо користувач тримає у руках предмет, та промінь нічого не перетинає, то предмет відкріпляється від гравця та падає на підлогу. Якщо ж перед гравцем знаходиться пустий стіл, то предмет переміщається на нього, а в самому столі оновлюється змінна, що зберігає предмет на ньому. Відповідно до рисунку 2.3 на зайнятий стіл неможливо нічого помістити.

Більш цікавою є функція розміщення страв на підносі для видачі замовлень. Піднос може містити до трьох предметів, які закріпляються на ньому та зберігаються у відповідній змінній класу Plate. Єдиною складністю в цій задачі є розвертання об’єктів на підносі разом із ним без зміни їх порядку. Рішення цієї проблеми показано на рисунку 3.20.

```
        if(this.plate.direction.x!=0&&this.plate.direction.z!=0){
                if(this.plate.food[0])          this.plate.food[0].position          =          new
CANNON.Vec3(this.plate.location[0]-this.plate.direction.z*1.2,          this.plate.location[1]+0.2,
this.plate.location[2]+this.plate.direction.x*0.8) ;
                if(this.plate.food[1])          this.plate.food[1].position          =          new
CANNON.Vec3(this.plate.location[0], this.plate.location[1]+0.2, this.plate.location[2]) ;
                if(this.plate.food[2])          this.plate.food[2].position          =          new
CANNON.Vec3(this.plate.location[0]+this.plate.direction.z*0.8,          this.plate.location[1]+0.2,
this.plate.location[2]-this.plate.direction.x*1.2) ;
              } else{
                if(this.plate.food[0])          this.plate.food[0].position          =          new
CANNON.Vec3(this.plate.location[0]+this.plate.direction.z,          this.plate.location[1]+0.2,
this.plate.location[2]+this.plate.direction.x) ;
                if(this.plate.food[1])          this.plate.food[1].position          =          new
CANNON.Vec3(this.plate.location[0], this.plate.location[1]+0.2, this.plate.location[2]) ;
                if(this.plate.food[2])          this.plate.food[2].position          =          new
CANNON.Vec3(this.plate.location[0]-this.plate.direction.z,          this.plate.location[1]+0.2,
this.plate.location[2]-this.plate.direction.x) ;
              }
```

Рисунок 3.20 – Розміщення та обертання предметів на підносі

Після розробки всіх вище зазначених функцій, необхідно протестувати коректність їх роботи. На рисунках 3.21 та 3.22 показано приклади роботи вебгри та працездатності описаних функцій.

Таким чином було розроблено головні механіки гри. Далі необхідно

створити один з головних механізмів – надходження та перевірка замовлень.

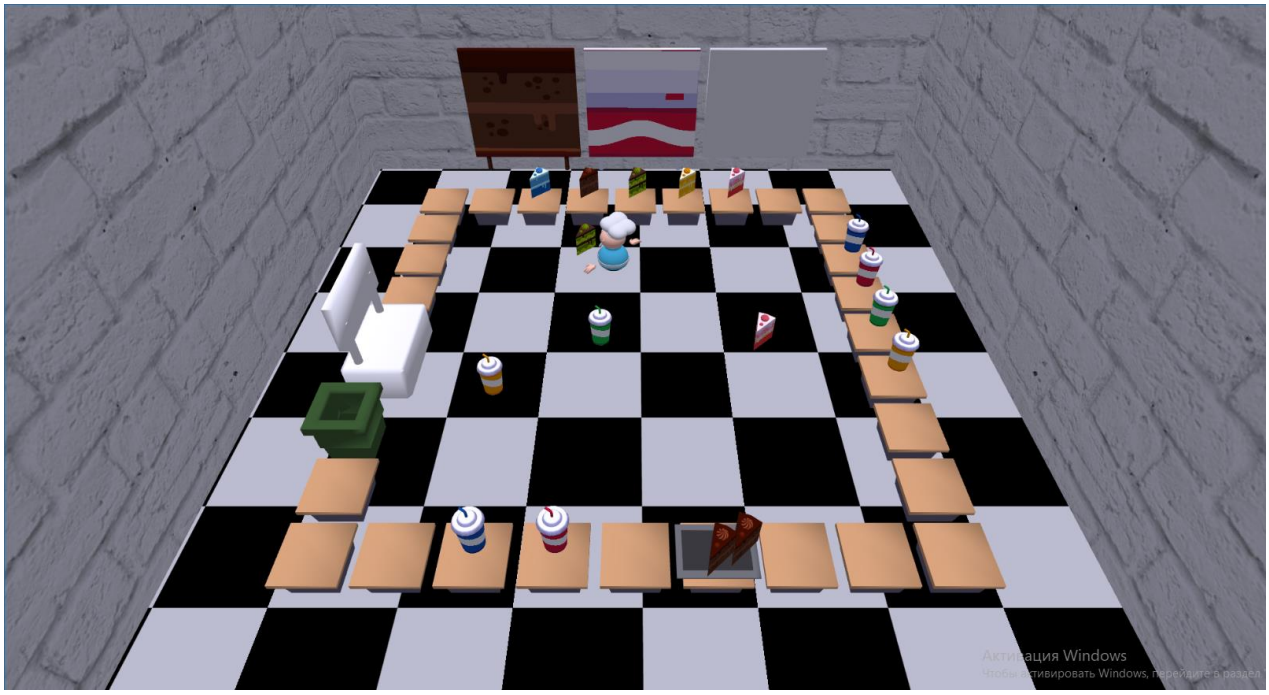

Рисунок 3.21 – Приклад роботи гри

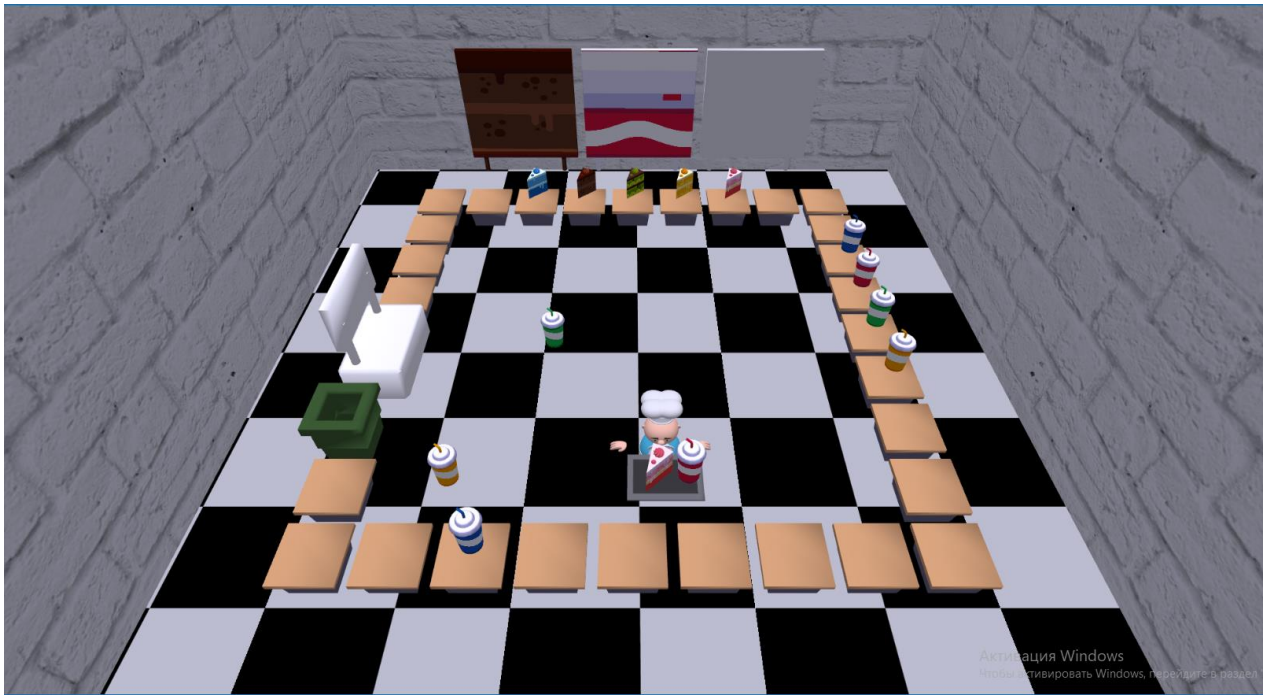

Рисунок 3.22 – Приклад роботи гри

Для реалізації замовлень та самої стійки замовлень, а також стендів, що показують замовлення, розробляється клас Order. Екземпляр цього класу є

схожим на екземпляр класу Kitchen, та при спробі користувачем покласти на нього піднос рахується як звичайний стіл. Однак при взаємодії з екземпляром класу Order та маючи піднос в руках, об’єкт підносу зберігається у класі замовлення для подальшої обробки.

Щоб обробляти замовлення, в першу чергу необхідно його створити. Функція на рисунку 3.23 показує, як генерується випадкове замовлення. Спочатку обирається випадкове число від 1 до 3, щоб визначити скільки страв буде у замовленні. Потім випадково зі списку існуючих страв обирається необхідна їх кількість та зберігається у відповідній змінній екземпляру класу Order. В той самий час до стендів, що відображатимуть замовлення, додається або текстура обраної страви, або біла текстура для позначення відсутності страви.

```
function MakeOrder(obj){
   var count = Math.floor(Math.random() * 3) + 1;

   for(var i=0; i<count; i++){
      var item = Math.floor(Math.random() * obj.food_names.length);
      obj.order.push(obj.food_names[item]);
   }

   for(var i=0; i<3; i++){
      if(obj.order[i]){
         for(var o in obj.stage.stageObjects){
            if(obj.stage.stageObjects[o].name == obj.order[i]){
               obj.boards[i].textureIndex = obj.stage.stageObjects[o].textureIndex;
            }
         }
      }
      else obj.boards[i].textureIndex = obj.clear_texture;

   }
}
```

Рисунок 3.23 – Створення замовлення

Коли замовлення створене, необхідно дочекатися, коли гравець поставить на стійку піднос із предметами. Після цього потрібно перевірити,

які саме страви знаходяться на підносі, та чи збігаються вони з замовленими. Для цього об’єкти на підносі перебираються та порівнюються зі стравами у замовленні. Якщо всі предмети збігаються, то замовлення вважається виконаним, всі предмети на підносі видаляються із ігрового світу для економії пам’яті комп’ютера, піднос повертається на своє початкове місце, а гра генерує нове замовлення. Однак, якщо замовлення виконано не вірно, піднос містить неправильні, зайві або не всі страви, то предмети все одно видаляються, проте замовлення не змінюється і його необхідно виконати з початку. Реалізацію такої перевірки продемонстровано на рисунку 3.24.

```
var order=[];
for(var i=0; i<this.order.length; i++){
order.push(this.order[i]);
}
if(this.plate.food.length > 0){
for(var i=0; i<this.plate.food.length; i++){
for(var o in this.stage.stageObjects){
if(this.stage.stageObjects[o].rigidBody == this.plate.food[i]){

if(order.includes(this.stage.stageObjects[o].name)){
const index =  order.indexOf(this.stage.stageObjects[o].name);
order.splice(index, 1);
}
const index =  this.stage.stageObjects.indexOf(this.stage.stageObjects[o]);
console.log(this.stage.stageObjects[o]);
this.stage.stageObjects.splice(index, 1);
break;
}
}
}
if(this.order.length == this.plate.food.length && order.length == 0) this.order = [];
this.plate.food = [];
}
```

Рисунок 3.24 – Перевірка правильності виконаного замовлення

Таким чином у грі відбувається нескінченний цикл надходження та виконання замовлень. Останнім кроком у реалізації вебгри є додавання смітника, щоб можна було викинуті зайві предмети або неправильно виконане

замовлення на підносі.

Для цього створюється клас Bin, який працює схожим чином із класом Order. Він приймає в себе страву, що тримає гравець, або масив страв що знаходяться на підносі. Після цього виконується код, що видаляє отримані об’єкти з ігрового світу, як показано на рисунку 3.25. При цьому при викиданні їжі з підноса, він не повертається на своє попереднє місце, а залишається в руках у гравця.

```
if(this.items){
    if(this.items.length > 0){
        for(var i=0; i<this.items.length; i++){
            for(var o in this.stage.stageObjects){
                if(this.stage.stageObjects[o].rigidBody == this.items[i]){

                    const index =  this.stage.stageObjects.indexOf(this.stage.stageObjects[o]);
                    this.stage.stageObjects.splice(index, 1);
                    delete this.items[i];
                    break;
                }
            }
        }
        this.items = [];
    }
}
```

Рисунок 3.25 – Видалення об’єктів з ігрового світу

Таким чином було реалізовано всі необхідні функції гри та можливості користувача. Було зроблено все необхідне для роботи вебгри та ігрового процесу, а також протестована працездатність гри.

На рисунку 3.26 показано фінальний вигляд гри та приклад тримання підносу з правильним замовленням на ньому.

На рисунку 3.27 демонструється поява нового замовлення, коли попереднє виконано вірно. Доказом працездатності виступає один об’єкт, що на обох зображеннях знаходиться на одному й тому самому місці.

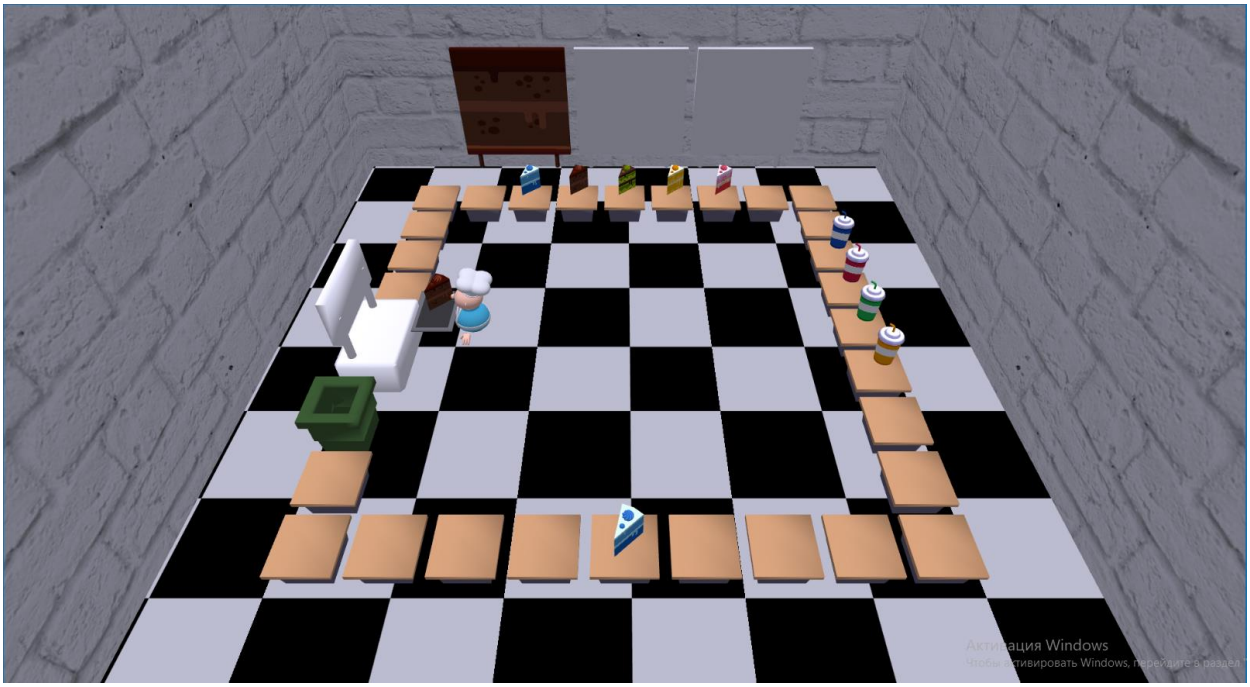

Рисунок 3.26 – Тестування гри

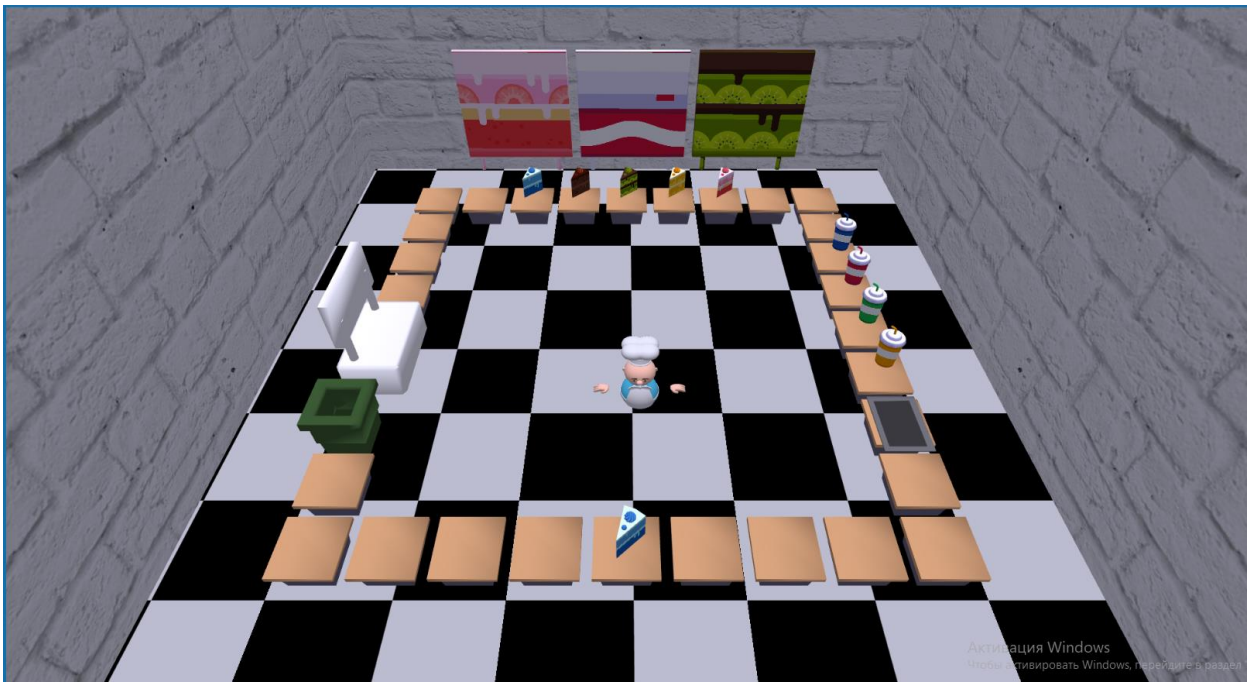

Рисунок 3.27 – Тестування гри

Таким чином на рисунках 3.26 та 3.27 показано приклад роботи гри та її фінальний зовнішній вигляд. Окрім всього функціоналу гри та ігрового рушія, на сторінку гри додається блок з правилами гри, а сама сторінка стилізується за допомогою CSS (реалізація цих пунктів наведена у додатку А).

3.3 Висновки до розділу 3

В розділі 3 було показано реалізацію ігрового рушія та вебгри на його основі з виконанням усіх вимог зазначених у розділі 2. Було наведено реалізацію основних функцій застосунку та описано їх призначення і принципи роботи. Також було проведено тестування розробленого додатку та наведено відповідні зображення з демонстрацією працездатності програми.

ВИСНОВКИ

В результаті роботи було досліджено процес розробки ігрового рушія засобами WebGL та JavaScript, реалізовано відповідний програмний застосунок, а також створено вебгру на його основі.

Спочатку було проаналізовано існуючі технології для розробки вебігор із 3D-графікою, наведено їх характеристику та аргументований вибір засобів для розробки застосунку. Після цього було визначено функціональні вимоги до ігрового рушія та вебгри, а також можливості користувача.

Наступним кроком було проєктування майбутніх частин програми, а саме ігрового рушія та вебгри на його основі. Спираючись на визначені раніше вимоги було розроблено структури класів обох частин проєкту, після чого відображено їх на UML-діаграмах класів. Окрім того було розроблено діаграму активності, щоб зрозуміло та наочно описати процес гри там можливості користувача у ній.

Останнім етапом була програмна реалізація застосунку, протягом якого, відповідно до розроблених раніше діаграм, було створено всі необхідні класи та задовільнено вимоги до проєкту, основними з яких є завантаження та відтворення 3D-графіки у браузері, створення фізичного світу та розробка ігрового процесу.

Таким чином було досягнута мета роботи – реалізація ігрового рушія засобами WebGL та JavaScript та розробка гри на його основі. Був аргументований вибір інструментів, спроєктовано застосунок, після чого розроблено відповідний програмний продукт.

ПЕРЕЛІК ПОСИЛАНЬ

1. Sumeet A. WebGL Game Development. Birmingham : Packt Publishing, 2014. 397 p.

2. Півняк Г. Г., Бусигін Б. С., Дівізінюк М. М. та ін. Тлумачний словник з інформатики. Дніпро : Нац. гірнич. ун-т, 2010. 600 с.

3. Patrick G. Henriette B. Philippe B. UML 2.0 in Action. Birmingham : Packt Publishing, 2005. 229 p.

4. Cannon. Github. URL: https://schteppe.github.io/cannon.js/docs/ (дата звернення: 05.04.2023).

5. WebGL 2.0 Specification. The Khronos Group Inc. URL: https://registry.khronos.org/webgl/specs/latest/2.0/ (дата звернення: 01.03.2023).

6. JavaScript documentation. DevDocs API Documentation. URL: https://devdocs.io/javascript/ (дата звернення: 01.03.2023).

7. Blender 3.5 Reference Manual. Blender Documentation. URL: https://docs.blender.org/manual/en/latest/ (дата звернення: 25.03.2023).

8. Спиця О. Г., Зіновєєв І. В., Манько Н. І.-В. Лінійна алгебра та аналітична геометрія : навчальний посібник для здобувачів ступеня вищої освіти бакалавра спеціальності 121 «Інженерія програмного забезпечення» освітньо-професійної програми «Програмна інженерія». Запоріжжя : ЗНУ, 2022. 96 с.

9. Matsuda K., Lea R. WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL. Michigan : Pearson Education, 2013. 516 p.

ДОДАТОК А

Код ігрового рушія

```
<html>

<head>
<title>Cooking</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" href="style.css">

<script type="text/javascript" src="js/gl-matrix-min.js"></script>
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript" src="js/webgl-utils.js"></script>
<script src="./js/cannon.js"></script>

<script type="text/javascript" src="primitive/Face.js"></script>
<script type="text/javascript" src="primitive/Geometry.js"></script>
<script type="text/javascript" src="primitive/parseJSON.js"></script>
<script type="text/javascript" src="primitive/utils.js"></script>
<script type="text/javascript" src="primitive/StageObject.js"></script>
<script type="text/javascript" src="primitive/Light.js"></script>
<script type="text/javascript" src="primitive/Lights.js"></script>

<script type="text/javascript" src="primitive/game/Chef.js"></script>
<script type="text/javascript" src="primitive/game/Floor.js"></script>
<script type="text/javascript" src="primitive/game/Kitchen.js"></script>
<script type="text/javascript" src="primitive/game/Food.js"></script>
<script type="text/javascript" src="primitive/game/Plate.js"></script>
<script type="text/javascript" src="primitive/game/Order.js"></script>
<script type="text/javascript" src="primitive/game/Bin.js"></script>

<script type="text/javascript" src="primitive/Stage.js"></script>

<script type="text/javascript" src="primitive/camera.js"></script>

<script id="shader-fs" type="x-shader/x-fragment">
   precision mediump float;
   varying vec3 transformedNormal;
   varying vec3 vertexPos;

   uniform vec3 uAmbientColor;
   uniform vec3 uLightingPosition;
   uniform vec3 uDirectionalColor;
   uniform vec3 uSpecularColor;

   uniform vec3 materialDiffuseColor;
   uniform vec3 materialAmbientColor;
   uniform vec3 materialSpecularColor;
   uniform sampler2D uSampler;
   varying highp vec2 vTextureCoord;
   uniform bool hasTexture;
```

```
            void main(void)  {
            vec3 normal=normalize(transformedNormal);
            vec3 eyeVector=normalize(-vertexPos);
            vec3 lightDirection = normalize(uLightingPosition);
            float specular = 0.0;

            float directionalLightWeighting = max(dot(normal, -lightDirection), 0.0);
            if(directionalLightWeighting>0.0)
            {
            vec3 halfDir = normalize(-lightDirection + eyeVector);
            float specAngle = max(dot(halfDir, normal), 0.0);
            specular = pow(specAngle, 4.0);
            }
            if(hasTexture){
            vec3 iColor = uAmbientColor+ uDirectionalColor*
directionalLightWeighting+uSpecularColor*materialSpecularColor*specular;
            vec4 tColor=texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));

            gl_FragColor = vec4(iColor, 1.0)*texture2D(uSampler, vec2(vTextureCoord.s,
vTextureCoord.t));
            }
            else{
            vec3 iColor = uAmbientColor*materialAmbientColor+ uDirectionalColor
*materialDiffuseColor * directionalLightWeighting+uSpecularColor*materialSpecularColor*specular;

            gl_FragColor = vec4(iColor, 1.0);
            }
            }
        </script>


        <script id="shader-vs" type="x-shader/x-vertex">
            attribute vec3 aVertexPosition;
            attribute vec3 aVertexNormal;
            uniform mat4 mVMatrix;
            uniform mat4 pMatrix;
            uniform mat4 nMatrix;
            varying vec3 transformedNormal;
            varying vec3 vertexPos;
            attribute vec2 aTextureCoord;
            varying highp vec2 vTextureCoord;
            uniform bool hasTexture;
            void main(void) {
            vec4 vertexPos4 =  mVMatrix * vec4(aVertexPosition, 1.0);
            vertexPos = vertexPos4.xyz;
            transformedNormal = vec3(nMatrix * vec4(aVertexNormal,1.0));
            gl_Position= pMatrix *vertexPos4;
             if(hasTexture){
              vTextureCoord = aTextureCoord;
             }
            }
        </script>


        <script type="text/javascript">
            var canvas;
            var gl;
```

```
var shaderProgram;
var mvMatrix = mat4.create();
var pMatrix = mat4.create();
var stage;
var textureCount=-1;

var move=0, plate=0, food = [], boards = [];
var food_names=["blue_cake", "chocolate_cake", "qiwi_cake", "fruit_cake",
"strawberry_cake", "blue_cola", "red_cola", "green_cola", "yellow_cola"];
var matrixStack = [];
var textureList=[];
var cam;
var tables=[];
var nextitemIndex=0;
var system=0;
var rate=30; //Animation should fire after every 300
var MINIMUM_FRAME_RATE=30;
var lastFrameTime = undefined;
var cyclesLeftOver;
var currentTime;
var elapsedTime;
var clampToEdge=true;

function pushMatrix() {
   var copy = mat4.create();
   mat4.copy(copy,mvMatrix);
   matrixStack.push(copy);
}

function popMatrix() {
   if (matrixStack.length == 0) {
      throw "Invalid popMatrix!";
   }
   mvMatrix = matrixStack.pop();
}
function initCannon() {
   system = new CANNON.World();
   system.gravity.set(0,  -9.8, 0); // m/s²
}
async function start() {
   canvas = document.getElementById("game");
   lastFrameTime=(new Date).getTime();

   initCannon();
   initGL(canvas);

   initShaders();

   stage=new Stage(gl);
   gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
   cam=new Camera();
   cam.setPosition(new Array(0.0, 20.0, 20.0));
   cam.setLookAtPoint(vec3.fromValues(0.0, 0.0, 0.0));

   initScene();

   await loadStageObject("model/obj/wall.json",[-30.0,0.0,0.0],0,0,0);
```

```
        await loadStageObject("model/obj/wall.json",[0.0,0.0,-30.0],0,Math.PI/2,0);
        await loadStageObject("model/obj/wall.json",[30.0,0.0,0.0],0,0,0);

        await loadStageObject("model/obj/board.json",[-17.0,0.0,-30.0],0.0,-Math.PI/2,0.0);
        await loadStageObject("model/obj/board.json",[-4.0,0.0,-30.0],0.0,-Math.PI/2,0.0);
        await loadStageObject("model/obj/board.json",[9.0,0.0,-30.0],0.0,-Math.PI/2,0.0);

        await loadStageObject("model/obj/table.json",[-20.0,0.0,-20.0],0.0,0.0,0.0);
        await loadStageObject("model/obj/floor.json",[0.0,0.0,0.0],0.0,0.0,0.0);

        await loadStageObject("model/obj/red_cola.json",[0.0,0.0,0.0],0.0,0.0,0.0);
        await loadStageObject("model/obj/blue_cola.json",[0.0,0.0,0.0],0.0,0.0,0.0);
        await loadStageObject("model/obj/yellow_cola.json",[0.0,0.0,0.0],0.0,0.0,0.0);
        await loadStageObject("model/obj/green_cola.json",[0.0,0.0,0.0],0.0,0.0,0.0);
        await loadStageObject("model/obj/blue_cake.json",[0.0,0.0,0.0],0.0,0.0,0.0);
        await loadStageObject("model/obj/qiwi_cake.json",[0.0,0.0,0.0],0.0,0.0,0.0);
        await loadStageObject("model/obj/chocolate_cake.json",[0.0,0.0,0.0],0.0,0.0,0.0);
        await loadStageObject("model/obj/fruit_cake.json",[0.0,0.0,0.0],0.0,0.0,0.0);
        await loadStageObject("model/obj/strawberry_cake.json",[0.0,0.0,0.0],0.0,0.0,0.0);



        await loadStageObject("model/obj/plate.json",[0.0,0.0,0.0],0.0,Math.PI/2,0.0);
        await loadStageObject("model/obj/order.json",[-20.0,2.0,3.0],0.0,0.0,0.0);
        await loadStageObject("model/obj/bin.json",[-20.0,2.0,5.0],0.0,0.0,0.0);
        await loadStageObject("model/obj/chef.json",[0.0,0.0,0.0],0.0,0.0,0.0);
    }


    async function loadStageObject(url,location,rotationX,rotationY,rotationZ){
        await $.getJSON(url,function(data){
            var nameOfObject=data.metadata.sourceFile.split(".")[0];
            var stageObject=null;

            if(nameOfObject=="chef"){
                stageObject=new Chef();
                stageObject.document = document;
                stageObject.tables = tables;
                move = stageObject;
            }
            else if(nameOfObject=="floor"){
                stageObject=new Floor();
            }
            else if(nameOfObject=="table"){
                stageObject=new Kitchen();
            }
            else if(food_names.includes(nameOfObject)){
                stageObject=new Food();


                var positions = [3,5,7,8,9,17,19,21,23];
                var num = positions[food_names.indexOf(nameOfObject)];
                food[food_names.indexOf(nameOfObject)] = stageObject;
                console.log(num)
                tables[num].food = stageObject;
                stageObject.place = vec3.fromValues(tables[num].location[0],
tables[num].location[1]+0.1, tables[num].location[2]);
```

```
                }
                else if(nameOfObject=="plate"){
                    stageObject = new Plate();
                    var num = 25;
                    tables[num].food = stageObject;
                    stageObject.place = vec3.fromValues(tables[num].location[0],
tables[num].location[1]+0.2, tables[num].location[2]);
                }
                else if(nameOfObject=="order"){
                    stageObject=new Order();
                    tables.push(stageObject);
                    stageObject.stage = stage;
                    stageObject.tables = tables;
                    stageObject.food_names = food_names;
                    stageObject.boards = boards;
                }
                else if(nameOfObject=="bin"){
                    stageObject=new Bin();
                    tables.push(stageObject);
                    stageObject.stage = stage;
                    stageObject.tables = tables;
                }
                else if(nameOfObject=="board"){
                    stageObject = new StageObject();
                    boards.push(stageObject);
                }
                else{
                    stageObject=new StageObject();
                }
                //stageObject.camera=cam;

                stageObject.loadObject(data);
                loadTexture(stageObject,false);
                addStageObject(stageObject,location,rotationX,rotationY,rotationZ);
                stageObject.system=system;
                if(stageObject.name!="board"&&stageObject.name!="wall")
stageObject.initializePhysics();

            });

        }
        function loadTexture(stageObject,bool){
            if(stageObject.materialFile!=null){
                if((textureList[stageObject.materialFile.trim()]===undefined)){
                    var currentDate=new Date();
                    var textureIndex=currentDate.getMilliseconds()*Math.random();
                    stageObject.textureIndex=textureIndex;
                    textureList[stageObject.materialFile.trim()]=textureIndex;

                    initTextures(stageObject.materialFile.trim(),textureIndex,bool);
                }
                else{
                    stageObject.textureIndex=textureList[stageObject.materialFile.trim()]

                }
            }
        }
```

```
        function initTextures(fileName,textureCount,clampToEdge){
            var image = new Image();
            image.onload = function() { stage.addTexture(textureCount,fileName,image,clampToEdge); }

            image.src = "model/textures/"+fileName;
        }

        async function addStageObject(stageObject,location,rotationX,rotationY,rotationZ){

            cloneObjects(stageObject);
            stageObject.location=location;
            stageObject.rotationX=rotationX;
            stageObject.rotationY=rotationY;
            stageObject.rotationZ=rotationZ;
            stage.addModel(stageObject);
        }
        function cloneObjects(stageObject){
            if(stageObject.name=="table"){
                tables.push(stageObject);
                stageObject.chef=move;

                for(var i = -20; i<=20; i+=40){
                    for(var j=-20; j<=20; j+=5){
                        if((i==-20 && j==-20) ) continue;
                        var table1=stageObject.clone();
                        table1.location=[j,0.0,i];
                        stage.addModel(table1);
                        table1.system = system;
                        table1.initializePhysics();
                        tables.push(table1);
                        table1.chef = move;

                        if(Math.abs(i)==Math.abs(j)|| (i==-20 && j==0)|| (i==-20 && j==5) || (i==-20 && j==10)) continue;
                        var table2=stageObject.clone();
                        table2.location=[i,0.0,j];
                        stage.addModel(table2);
                        table2.system = system;
                        table2.initializePhysics();
                        tables.push(table2);
                        table2.chef = move;
                    }
                }
            }
            if(food_names.includes(stageObject.name)){
                var items=[];
                for(var j=0;j<3;++j){
                    var item=stageObject.clone();
                    items.push(item);
                    item.system=system;
                    item.visible = false;

                    stage.addModel(item);
                }
                stageObject.stack = items;
            }
```

```
}

function initScene(){

        gl.clearColor(1.0, 1.0, 1.0, 1.0);
        gl.enable(gl.DEPTH_TEST);
        mat4.identity(mvMatrix);

        tick();

}
function tick(){
    requestAnimFrame(tick);

    animate();

}
function animate(){
    currentTime = (new Date).getTime();
    elapsedTime = currentTime - lastFrameTime;
    if (elapsedTime < rate) return;


    var updateIterations = Math.floor(elapsedTime / MINIMUM_FRAME_RATE);

    system.step( elapsedTime / 1000);
    while(updateIterations > 0){

       try{
        drawScene();
       }
       catch(e){
        console.log(e);
       }

        updateIterations -= 1;
    }
    lastFrameTime = (new Date).getTime();


}

function initGL(canvas) {

        var names = ["webgl", "experimental-webgl", "webkit-3d", "moz-webgl"];

        for (var i = 0; i < names.length; ++i) {
            try {
                gl = canvas.getContext(names[i]);
            }
            catch (e) { }
            if (gl) {
                break;
            }
        }
        if (gl == null) {
            alert("Could not initialise WebGL");
```

```
            return null;
        }

        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
}


function getShader(gl, id) {
    var shaderScript = document.getElementById(id);
    if (!shaderScript) {
        return null;
    }

    var str = "";
    var k = shaderScript.firstChild;
    while (k) {
        if (k.nodeType == 3) {
            str += k.textContent;
        }
        k = k.nextSibling;
    }

    var shader;
    if (shaderScript.type == "x-shader/x-fragment") {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    } else if (shaderScript.type == "x-shader/x-vertex") {
        shader = gl.createShader(gl.VERTEX_SHADER);
    } else {
        return null;
    }

    gl.shaderSource(shader, str);
    gl.compileShader(shader);

    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        alert(gl.getShaderInfoLog(shader));
        return null;
    }

    return shader;
}

function initShaders() {
    var fragmentShader = getShader(gl, "shader-fs");
    var vertexShader = getShader(gl, "shader-vs");

    shaderProgram = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);

    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
        alert("Could not initialise shaders");
    }

    gl.useProgram(shaderProgram);
```

```
            shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram,
"aVertexPosition");
            gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
            shaderProgram.vertexNormalAttribute = gl.getAttribLocation(shaderProgram,
"aVertexNormal");
            gl.enableVertexAttribArray(shaderProgram.vertexNormalAttribute);

            shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "pMatrix");
            shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "mVMatrix");
            shaderProgram.nMatrixUniform = gl.getUniformLocation(shaderProgram, "nMatrix");
            shaderProgram.ambientColorUniform = gl.getUniformLocation(shaderProgram,
"uAmbientColor");
            shaderProgram.specularColorUniform = gl.getUniformLocation(shaderProgram,
"uSpecularColor");
            shaderProgram.lightingDirectionUniform = gl.getUniformLocation(shaderProgram,
"uLightingPosition");
            shaderProgram.directionalColorUniform = gl.getUniformLocation(shaderProgram,
"uDirectionalColor");
            shaderProgram.materialDiffuseColor = gl.getUniformLocation(shaderProgram,
"materialDiffuseColor");
            shaderProgram.materialAmbientColor = gl.getUniformLocation(shaderProgram,
"materialAmbientColor");
            shaderProgram.materialSpecularColor = gl.getUniformLocation(shaderProgram,
"materialSpecularColor");
            shaderProgram.textureCoordAttribute = gl.getAttribLocation(shaderProgram,
"aTextureCoord");
            gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);
            shaderProgram.hasTexture = gl.getUniformLocation(shaderProgram, "hasTexture");
        }

        function calculateMvMatrix(){

            cam.apply(gl.viewportWidth / gl.viewportHeight);
            mat4.multiply(mvMatrix,mvMatrix,cam.viewMatrix);

        }
        function setMatrixUniforms() {
            gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, cam.projMatrix);
            gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
            var invertMatrix = mat4.create();
            mat4.invert(invertMatrix, mvMatrix);
            var normalMatrix=mat4.create();
            mat4.transpose(normalMatrix, invertMatrix)

            gl.uniformMatrix4fv(shaderProgram.nMatrixUniform, false, normalMatrix);
        }
        function setLightUniform(){

            gl.uniform3f(shaderProgram.ambientColorUniform,0.5,0.5,0.55);
            gl.uniform3f(shaderProgram.specularColorUniform,1.0,1.0,1.0);
            var lightingDirection = [0,-1.25,-1.25];
            gl.uniform3fv(shaderProgram.lightingDirectionUniform, lightingDirection);
            gl.uniform3f(shaderProgram.directionalColorUniform,0.5,0.5,0.55);
        }

        function drawScene() {
```

```
            gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
            gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);


            setLightUniform();
            pushMatrix();
            calculateMvMatrix();
            for(var i=0;i<stage.stageObjects.length;i++){
             if(stage.stageObjects[i].visible==true){

            pushMatrix();

            if(stage.stageObjects[i]) stage.stageObjects[i].update();
            if(!stage.stageObjects[i]) i-=5;
            if(stage.stageObjects[i]){

            mat4.multiply(mvMatrix,mvMatrix,stage.stageObjects[i].modelMatrix);

            setMatrixUniforms();
            gl.bindBuffer(gl.ARRAY_BUFFER, stage.stageObjects[i].vbo);
            gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
stage.stageObjects[i].vbo.itemSize, gl.FLOAT, false, 0, 0);
            gl.bindBuffer(gl.ARRAY_BUFFER,stage.stageObjects[i].nbo);
            gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
stage.stageObjects[i].nbo.itemSize, gl.FLOAT, false, 0, 0);

gl.uniform3f(shaderProgram.materialDiffuseColor,stage.stageObjects[i].diffuseColor[0],stage.stageObjec
ts[i].diffuseColor[1],stage.stageObjects[i].diffuseColor[2]);

gl.uniform3f(shaderProgram.materialAmbientColor,stage.stageObjects[i].ambientColor[0],stage.stageObj
ects[i].ambientColor[1],stage.stageObjects[i].ambientColor[2]);

gl.uniform3f(shaderProgram.materialSpecularColor,stage.stageObjects[i].specularColor[0],stage.stageObj
ects[i].specularColor[1],stage.stageObjects[i].specularColor[2]);
            if(stage.stageObjects[i].materialFile!=null){
               gl.uniform1i(shaderProgram.hasTexture,1);
               try{
               gl.activeTexture(gl.TEXTURE0);
               gl.bindTexture(gl.TEXTURE_2D,
stage.textures[stage.stageObjects[i].textureIndex].texture);
               gl.uniform1i(gl.getUniformLocation(shaderProgram, "uSampler"), 0);
               gl.bindBuffer(gl.ARRAY_BUFFER, stage.stageObjects[i].verticesTextureBuffer);
               gl.vertexAttribPointer(shaderProgram.textureCoordAttribute, 2, gl.FLOAT, false, 0, 0);
               }
               catch(e){
               }
            }
            else{
               gl.uniform1i(shaderProgram.hasTexture,0);

            }
               gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, stage.stageObjects[i].ibo);

               gl.drawElements(gl.TRIANGLES,          stage.stageObjects[i].geometry.indices.length,
gl.UNSIGNED_SHORT,0);
            popMatrix();
```

```
            }
            else console.log(stage.stageObjects[i]+" "+i+" "+stage.stageObjects.length);
           }}
           popMatrix();

           if(food){
              for(var i=0; i<food.length; i++){
                 if(food[i]&&food[i].stack.length == 0) cloneObjects(food[i]);
              }
           }

        }
    </script>

    </head>

    <body onload="start();">
        <div class="flexbox">
            <canvas id="game" width="1980px" height="1080px"></canvas>
            <div id="rules">
                <h2>Правила гри</h2>
                <ul>
                    <li>Використовуйте WASD для керування персонажем</li>
                    <li>На стійках у кінці кухні відображається замовлення</li>
                    <li>На столах кухні лежать різні страви - замовлення показує, які саме страви хоче отримати клієнт</li>
                    <li>Використовуйте E щоб брати та класти предмети на столи або на підлогу</li>
                    <li>На одному з столів є піднос - на нього треба покласти обрані страви</li>
                    <li>Щоб виконати замовлення підійдіть з підносом до стійки видачі замовлень та натисніть E</li>
                    <li>Якщо замовлення виконано вірно - з'явиться нове замовлення, в іншому випадку замовлення доведеться переробити</li>
                    <li>Якщо на підносі або у руках є страви, що вам не потрібні - викиньте їх до смітника за допомогою E</li>
                    <li>Насолоджуйтеся грою!</li>
                </ul>
            </div>
        </div>

    </body>

    </html>


    #game {
        border: 15px double rgb(22, 111, 171);
        background-color:transparent;
        width:  85%;
        height: 85%;
        margin: 10px;
      }

    .flexbox{
        display: flex;
        flex-direction: column-reverse;
        align-items: center;
    }
```

```
#rules{
   font-family: Comic Sans MS, Comic Sans, cursive;
   font-size: x-large;
}

body{
   background-color:rgb(160, 232, 252);
}


Face = function ( a, b, c, normal, color, materialIndex ) {

   this.a = a;
   this.b = b;
   this.c = c;

   this.normal =  normal ;
   this.vertexNormals = [ ];

   this.vertexColors = color instanceof Array ? color : [];
   this.colorIndex = color;

   //this.vertexTangents = [];

   this.materialIndex = materialIndex !== undefined ? materialIndex : 0;



};
Face.prototype = {

   constructor: Face,
   clone: function () {

      var face = new Face(this.a, this.b, this.c );
      if(!(this.normal===undefined))
      face.normal=vec3.clone(this.normal );
      face.colorIndex= this.colorIndex;


      face.materialIndex = this.materialIndex;

      var i;

      for ( i = 0; i < this.vertexNormals.length; ++i){

         face.vertexNormals[ i ] = vec3.clone(this.vertexNormals[ i ]);
      }



      for ( i = 0; i<this.vertexColors.length;++i ){
         face.vertexColors[ i ] = this.vertexColors[ i ];

      }
```

```
        return face;

    }
};


Geometry = function () {

    this.vertices = [];
    this.verticesFromFile=[];
    this.uvsFromFile=[];

    this.uvs=[[]];
    this.colors = [];
    this.normals = [];
    this.indices=[];
    this.faces = [];
    this.faceUvs = [[]];
    this.faceVertexUvs = [[]];
    this.textureCoordinates=[];
    this.materials=[];

};

    Geometry.prototype.indicesFromFaces=function (){

        for(var i=0;i<this.faces.length;++i){
            this.indices.push(this.faces[i].a);
            this.indices.push(this.faces[i].b);
            this.indices.push(this.faces[i].c);

        }

    }
Geometry.prototype.meshFromFaces=function(){
        var redundantVertices=[];
        var newIndex=0;
        this.indices=[];
        this.normals=[];

        for(var i=0;i<this.faces.length;i=i+1){
            var aVertices=["a","b","c"];
            for(var j=0;j<3;++j){
                var aVertex=aVertices[j];

                if(!(this.faces[i].normal==undefined)&&this.faces[i].vertexNormals.length==0){
                    this.faces[i].vertexNormals[aVertex]=vec3.clone(this.faces[i].normal);
                }

                redundantVertices.push(this.verticesFromFile[this.faces[i][aVertex]*3]);
                redundantVertices.push(this.verticesFromFile[this.faces[i][aVertex]*3+1]);
                redundantVertices.push(this.verticesFromFile[this.faces[i][aVertex]*3+2]);
                this.indices.push(newIndex);
                this.normals.push(this.faces[i].vertexNormals[aVertex][0]);
                this.normals.push(this.faces[i].vertexNormals[aVertex][1]);
                this.normals.push(this.faces[i].vertexNormals[aVertex][2]);
                newIndex=newIndex+1;
```

```
        }

    }

    this.vertices=redundantVertices;
}
Geometry.prototype.verticesFromFaceUvs=function(vertices,uvs,materialIndex){

    var vertexVectors=[];
    var redundantVertexVectors=[];

    var vertexCovered=[];
    for(var i=0;i<vertices.length;i=i+3){
        var vectorA=vec3.fromValues(vertices[i],vertices[i+1],vertices[i+2]);

        vertexVectors.push(vectorA);
    }

    var count=0;
    for(var i=0;i<this.faceVertexUvs[materialIndex].length;++i)
    {
        var face=this.faces[i];
        var textureIndices=this.faceVertexUvs[materialIndex][i];
        var aVertexIndices=["a","b","c"];
        for(var vertexIndex=0;vertexIndex<aVertexIndices.length;++vertexIndex){
            var aVertexIndex=aVertexIndices[vertexIndex];


if(redundantVertexVectors[textureIndices[aVertexIndex]]==face[aVertexIndex]||redundantVertexVectors
[textureIndices[aVertexIndex]]===undefined){

                if(face[aVertexIndex]==undefined) console.log("error" + aVertexIndex);

                redundantVertexVectors[textureIndices[aVertexIndex]]=face[aVertexIndex];
                face[aVertexIndex]=textureIndices[aVertexIndex];
            }

            else{

                uvs[materialIndex].push(uvs[materialIndex][textureIndices[aVertexIndex]*2]);
                uvs[materialIndex].push(uvs[materialIndex][textureIndices[aVertexIndex]*2+1]);
                var newIndex=Math.floor(uvs[materialIndex].length/2)-1;
                if(face[aVertexIndex]==undefined) console.log("error" + aVertexIndex);
                redundantVertexVectors[newIndex]=face[aVertexIndex];
                face[aVertexIndex]=newIndex;
                textureIndices[aVertexIndex]=newIndex
            }
        }
    }

    for(var i=0;i<redundantVertexVectors.length;++i)
    {
        var vectorA=vertexVectors[redundantVertexVectors[i]];
        this.vertices.push(vectorA[0]);
        this.vertices.push(vectorA[1]);
        this.vertices.push(vectorA[2]);
    }
```

```
            this.uvs=uvs;
        }
    Geometry.prototype.morphedVertexNormalsFromObj=function(){
        var vertexVectors=[];
        var normalVectors=[];

        for(var i=0;i<this.faces.length;i=i+1){
            if(!(this.faces[i].normal==undefined)&&this.faces[i].vertexNormals.length==0){
                this.faces[i].vertexNormals["a"]=vec3.clone(this.faces[i].normal);
                this.faces[i].vertexNormals["b"]=vec3.clone(this.faces[i].normal);
                this.faces[i].vertexNormals["c"]=vec3.clone(this.faces[i].normal);
            }
            if(normalVectors[this.faces[i].a]===undefined)
            {
                normalVectors[this.faces[i].a]=vec3.clone(this.faces[i].vertexNormals["a"]);
            }
            else{

vec3.add(normalVectors[this.faces[i].a],normalVectors[this.faces[i].a],this.faces[i].vertexNormals["a"])
            }
            if(normalVectors[this.faces[i].b]===undefined){
                normalVectors[this.faces[i].b]=vec3.clone(this.faces[i].vertexNormals["b"]);
            }
            else{

vec3.add(normalVectors[this.faces[i].b],normalVectors[this.faces[i].b],this.faces[i].vertexNormals["b"])
            }
            if(normalVectors[this.faces[i].c]===undefined){
                normalVectors[this.faces[i].c]=vec3.clone(this.faces[i].vertexNormals["c"]);
            }
            else{

vec3.add(normalVectors[this.faces[i].c],normalVectors[this.faces[i].c],this.faces[i].vertexNormals["c"])
            }

        }
        this.normals=[];
        for(var j=0;j<normalVectors.length;j=j+1){
            vec3.normalize(normalVectors[j],normalVectors[j]);
            this.normals.push(normalVectors[j][0]);
            this.normals.push(normalVectors[j][1]);
            this.normals.push(normalVectors[j][2]);

        }


    }
    Geometry.prototype.calculateVertexNormals=function(){
        var vertexVectors=[];
        var normalVectors=[];

        var j;
        for(var i=0;i<this.vertices.length;i=i+3){
            var vector=vec3.fromValues(this.vertices[i],this.vertices[i+1],this.vertices[i+2]);
            var normal=vec3.create();//Intiliazed normal array
            normalVectors.push(normal);
            vertexVectors.push(vector);
```

```
            }
            try{
               for(j=0;j<this.indices.length;j=j+3)//Since we are using triads of indices to represent one
primitive
               {

                  //v1-v0
                  var vector1=vec3.create();
                  vec3.subtract(vector1,vertexVectors[this.indices[j+1]],vertexVectors[this.indices[j]]);
                  //v2-v1
                  var vector2=vec3.create();

vec3.subtract(vector2,vertexVectors[this.indices[j+2]],vertexVectors[this.indices[j+1]]);
                  var normal=vec3.create();
                  //cross product of two vector
                  vec3.cross(normal, vector1, vector2);
                  //Since the normal calculated from three vertices is same for all the three vertices(same
face/surface), the contribution from each normal to the corresponding vertex  is the same
                  vec3.add(normalVectors[this.indices[j]],normalVectors[this.indices[j]],normal);
                  vec3.add(normalVectors[this.indices[j+1]],normalVectors[this.indices[j+1]],normal);
                  vec3.add(normalVectors[this.indices[j+2]],normalVectors[this.indices[j+2]],normal);

               }
            }catch(e){
               alert(e);
            }
            for(var j=0;j<normalVectors.length;j=j+1){
               vec3.normalize(normalVectors[j],normalVectors[j]);
               this.normals.push(normalVectors[j][0]);
               this.normals.push(normalVectors[j][1]);
               this.normals.push(normalVectors[j][2]);

            }


         },
            Geometry.prototype.clone=function(){
            var geometry=new Geometry();
            var i,j;
            for(i=0;i<this.vertices.length;++i){
               geometry.vertices[i]=this.vertices[i];
            }
            for(i=0;i<this.faces.length;++i){
               geometry.faces[i]=this.faces[i].clone();
            }
            for(i=0;i<this.uvs.length;++i){
               for(j=0;j<this.uvs[i].length;++j) geometry.uvs[i][j]=this.uvs[i][j];
            }
            for(i=0;i<this.indices.length;++i){
               geometry.indices[i]=this.indices[i];
            }


            for(i=0;i<this.colors.length;++i){
               geometry.colors[i]=this.colors[i];
            }
            for(i=0;i<this.normals.length;++i){
```

```
                geometry.normals[i]=this.normals[i];
            }
            for(i=0;i<this.textureCoordinates.length;++i){
                geometry.textureCoordinates[i]=this.textureCoordinates[i];
            }
            for(i=0;i<this.faceUvs.length;++i){
                for(j=0;j<this.faceUvs[i].length;++j) geometry.faceUvs[i][j]=this.uvs[i][j];
            }
            for(i=0;i<this.faceVertexUvs.length;++i){
                for(j=0;j<this.faceVertexUvs[i].length;++j)
geometry.faceVertexUvs[i][j]=this.faceVertexUvs[i][j];
            }
            geometry.materials=jQuery.extend(true, {}, this.materials);
            return geometry;
        }


    function isBitSet( value, position ) {

        return value & ( 1 << position );

    }
    function parseJSON(data){

        var geometry=new Geometry();;
        var i, j, fi,

            offset, zLength, nVertices,

            colorIndex, normalIndex, uvIndex, materialIndex,

            type,
            isQuad,
            hasMaterial,
            hasFaceUv, hasFaceVertexUv,
            hasFaceNormal, hasFaceVertexNormal,
            hasFaceColor, hasFaceVertexColor,

            vertex, face, color, normal,

            uvLayer, uvs, u, v,

            faces = data.faces,
            vertices = data.vertices,
            normals = data.normals,
            colors = data.colors,
            nUvLayers = 0;
        geometry.verticesFromFile=data.vertices;
        // disregard empty arrays

        for ( i = 0; i < data.uvs.length; i++ ) {

            if ( data.uvs[ i ].length ) nUvLayers ++;

        }
```

```
for ( i = 0; i < nUvLayers; i++ ) {

   geometry.faceUvs[ i ] = [];
   geometry.faceVertexUvs[ i ] = [];

}


offset = 0;
zLength = faces.length;

while ( offset < zLength ) {

   type = faces[ offset ++ ];


   isQuad             = isBitSet( type, 0 );
   hasMaterial        = isBitSet( type, 1 );
   hasFaceUv          = isBitSet( type, 2 );
   hasFaceVertexUv    = isBitSet( type, 3 );
   hasFaceNormal      = isBitSet( type, 4 );
   hasFaceVertexNormal = isBitSet( type, 5 );
   hasFaceColor       = isBitSet( type, 6 );
   hasFaceVertexColor  = isBitSet( type, 7 );


   if ( isQuad ) {

      face = new Face();

      face.a = faces[ offset ++ ];
      face.b = faces[ offset ++ ];
      face.c = faces[ offset ++ ];
      face.d=faces[ offset ++ ];

      nVertices = 4;

   } else {

      face = new Face();

      face.a = faces[ offset ++ ];
      face.b = faces[ offset ++ ];
      face.c = faces[ offset ++ ];

      nVertices = 3;

   }

   if ( hasMaterial ) {

      materialIndex = faces[ offset ++ ];
      face.materialIndex = materialIndex;

   }

   //Just iterating and moving offset index forward. UV not relevant to this chapter.
```

```
fi = geometry.faces.length;

//Just iterating and moving offset index forward. UV not relevant to this chapter.

if ( hasFaceUv ) {

   for ( i = 0; i < nUvLayers; i++ ) {

      uvLayer = data.uvs[ i ];

      uvIndex = faces[ offset ++ ];



      geometry.faceUvs[ i ][ fi ] = uvIndex;

   }

}
//Just iterating and moving offset index forward. UV not relevant to this chapter.
if ( hasFaceVertexUv ) {

   for ( i = 0; i < nUvLayers; i++ ) {


      uvLayer = data.uvs[ i ];

      uvs = [];

      var aVertexIndices=["a","b","c","d"];
      for ( j = 0; j < nVertices; j ++ ) {

         uvIndex = faces[ offset ++ ];

         //u = uvLayer[ uvIndex * 2 ];
         //v = uvLayer[ uvIndex * 2 + 1 ];

         uvs[ aVertexIndices[j] ] = uvIndex;

      }

      geometry.faceVertexUvs[ i ][ fi ] = uvs;
   }

}


if ( hasFaceNormal ) {

   normalIndex = faces[ offset ++ ] * 3;
   normal = vec3.fromValues(normals[ normalIndex ++ ],normals[ normalIndex
++ ],normals[ normalIndex ]);
   face.normal = normal;

}

if ( hasFaceVertexNormal ) {
```

```
                var aVertices=["a","b","c","d"]
                for ( i = 0; i < nVertices; i++ ) {
                    var aVertex=aVertices[i];
                    normalIndex = faces[ offset ++ ] * 3;
                    normal = vec3.fromValues(normals[ normalIndex ++ ],normals[ normalIndex ++ ],normals[ normalIndex ]);
                    face.vertexNormals[aVertex]= normal;

                }
            }


            if ( hasFaceColor ) {

                colorIndex = faces[ offset ++ ];

                face.colorIndex = colorIndex;

            }


            if ( hasFaceVertexColor ) {

                for ( i = 0; i < nVertices; i++ ) {

                    colorIndex = faces[ offset ++ ];

                    face.vertexColors.push( colorIndex );

                }

            }

            geometry.faces.push( face );


        }
        geometry.materials=data.materials;
        geometry.verticesFromFaceUvs(data.vertices,data.uvs,0);

        geometry.indicesFromFaces();

        if(geometry.vertices.length==0){

            geometry.vertices=vertices;

        }
        if(data.normals.length>0){

            geometry.morphedVertexNormalsFromObj();
        }
        else{
            geometry.calculateVertexNormals();
        }
        //geometry.meshFromFaces();
```

```
    return geometry;
}


Camera =function ()
{
  // Raw Position Values
  this.left = vec3.fromValues(1.0, 0.0, 0.0); // Camera Left vector
  this.up = vec3.fromValues(0.0, 1.0, 0.0); // Camera Up vector
  this.dir = vec3.fromValues(0.0, 0.0, 1.0); // The direction its looking at
  this.pos = vec3.fromValues(0.0, 0.0, 0.0); // Camera eye position
  //this.projectionTransform = null;
  this.projMatrix;
  this.viewMatrix;

  this.fieldOfView = 55;
  this.nearClippingPlane = 0.1;
  this.farClippingPlane = 1500.0;
};
Camera.prototype.setPosition =function (newVec)
{
  this.pos=vec3.fromValues(newVec[0],newVec[1],newVec[2])

}
Camera.prototype.setLookAtPoint =function (newVec)
{

  if (isVectorEqual(this.pos, [0, 0, 0]) && isVectorEqual(newVec, [0, 0, 0]))
  {

  }
  else
  {
    // Figure out the direction of the point we are looking at.
    vec3.subtract(this.dir,newVec, this.pos);
     vec3.normalize(this.dir,this.dir);
    // Adjust the Up and Left vectors accordingly
    vec3.cross(this.left,vec3.fromValues(0, 1, 0), this.dir );
    vec3.normalize(this.left,this.left);
    vec3.cross(this.up,this.dir, this.left);
    vec3.normalize(this.up,this.up);
  }
}
Camera.prototype.apply =function (aspectRatio)
{
  var matView=mat4.create();
  var lookAtPosition=vec3.create();
  vec3.add(lookAtPosition,this.pos, this.dir);
  mat4.lookAt(matView, this.pos, lookAtPosition, this.up);
  mat4.translate(matView,matView,vec3.fromValues(-this.pos[0], -this.pos[1], -this.pos[2]));
  this.viewMatrix = matView;

  this.projMatrix=mat4.create();
  mat4.perspective(this.projMatrix, degToRadian(this.fieldOfView), aspectRatio,
this.nearClippingPlane, this.farClippingPlane)
}
```

```
Camera.prototype.getFarClippingPlane =function ()
{
 return this.farClippingPlane;
}

Camera.prototype.getFieldOfView =function ()
{
 return this.fieldOfView;
}

Camera.prototype.getLeft =function ()
{
 return vec3.clone(this.left);
}

Camera.prototype.getNearClippingPlane =function ()
{
 return this.nearClippingPlane;
}

Camera.prototype.getPosition =function ()
{
 return vec3.clone(this.pos);
}

Camera.prototype.getProjectionMatrix =function ()
{
 return mat4.clone(this.projMatrix);
}


Camera.prototype.getViewMatrix =function ()
{
 return mat4.clone(this.viewMatrix);
}

Camera.prototype.getUp =function ()
{
 return vec3.clone(this.up);
}

Camera.prototype.setFarClippingPlane =function (fcp)
{
 if (fcp > 0)
 {
  this.farClippingPlane = fcp;
 }
}

Camera.prototype.setFieldOfView =function (fov)
{
 if (fov > 0 && fov < 180)
 {
  this.fieldOfView = fov;
 }
}
```

```
Camera.prototype.setNearClippingPlane =function (ncp)
{
 if (ncp > 0)
 {
  this.nearClippingPlane = ncp;
 }
}

Camera.prototype.update =function (timeStep,linVel,angularVel)
{

 if (vec3.squaredLength(linVel)==0 && vec3.squaredLength(angularVel)==0) return false;

 if (vec3.squaredLength(linVel) > 0.0)
 {
  // Add a velocity to the position
  vec3.scale(velVec,velVec, timeStep);

  vec3.add(this.pos, velVec, this.pos);
 }

 if (vec3.squaredLength(angularVel) > 0.0)
 {
  // Apply some rotations to the orientation from the angular velocity
  this.pitch(angularVel[0] * timeStep);
  this.yaw(angularVel[1] * timeStep);
  this.roll(angularVel[2] * timeStep);
 }

 return true;

}

Stage = function (gl) {
   this.stageObjects=[];
   this.gl=gl;
   this.textures=new Object(); //This will have texture index,filename,loaded img object,and texture
object
};
Stage.prototype = {
   constructor: Stage,
   addModel:function(stageObject){

      if(!(this.gl===undefined))  {
         stageObject.createBuffers(this.gl);

      }
      this.stageObjects.push(stageObject);
   },
   getByName:function(name){
     var sceneObject=undefined;
     for(var i=0;i<this.stageObjects.length;++i){
        if(this.stageObjects[i].name==name){
         sceneObject=this.stageObjects[i];
         break;
        }
     }
```

```
        return sceneObject;
      },
      checkTexture:function(name){
        var flag=-1;
        for(var i=0;i<this.textures.length;++i){

          if(name==this.textures[i].fileName){
            flag=this.textures[i].index;
            break;
          }
        }
        return flag;
      },
      addTexture:function(index,name,img,clampToEdge){
        var texture=new Object();
        texture.fileName=name;
        texture.img=img;
        texture.index=index;
        texture.texture=this.gl.createTexture();
        this.gl.bindTexture(this.gl.TEXTURE_2D, texture.texture);

        this.gl.texImage2D(this.gl.TEXTURE_2D, 0, this.gl.RGBA, this.gl.RGBA,
this.gl.UNSIGNED_BYTE, img);
        this.gl.texParameteri(this.gl.TEXTURE_2D, this.gl.TEXTURE_MAG_FILTER,
this.gl.NEAREST);
        this.gl.texParameteri(this.gl.TEXTURE_2D, this.gl.TEXTURE_MIN_FILTER,
this.gl.NEAREST);
        if(clampToEdge){
          this.gl.texParameteri(this.gl.TEXTURE_2D, this.gl.TEXTURE_WRAP_S,
this.gl.CLAMP_TO_EDGE);
          this.gl.texParameteri(this.gl.TEXTURE_2D, this.gl.TEXTURE_WRAP_T,
this.gl.CLAMP_TO_EDGE);

        }
        this.gl.bindTexture(this.gl.TEXTURE_2D, null);
        this.textures[index]=texture;

      }
    };


    StageObject=function(){
      this.name="";
      this.geometry=new Geometry();
      this.location=vec3.fromValues(0,0,0);
      this.rotationX=0.0;
      this.rotationY=0.0;
      this.rotationZ=0.0;
      this.ibo=null;//Index buffer object
      this.vbo=null;//Buffer object for vertices
      this.nbo=null;//Buffer Object for normals
      this.diffuseColor=[1.0,1.0,1.0,1.0];
      this.ambientColor=[1.0, 1.0, 1.0];
      this.specularColor=[0.000000000000001, 0.0000000000000001, 0.0000000000000001];
      this.verticesTextureBuffer=null;
      this.textureIndex;
      this.materialFile;
```

```
        //this.camera=null;
        this.visible=true;
        this.modelMatrix=mat4.create();
        this.rigidBody=null;
        this.system=null;
        //this.callBack=null;
        //this.Keys = null;

    };
    StageObject.prototype.loadObject= function (data){

        this.geometry=parseJSON(data);


        this.name=data.metadata.sourceFile.split(".")[0];

        if(this.geometry.materials.length>0){
            try {
                if(!(this.geometry.materials[0].colorDiffuse===undefined))
                    this.diffuseColor=this.geometry.materials[0].colorDiffuse;

            } catch (error) {
                console.log(error);
            }
            if(!(this.geometry.materials[0].colorAmbient===undefined))
                this.ambientColor=this.geometry.materials[0].colorAmbient;
            if(!(this.geometry.materials[0].mapDiffuse===undefined)){
                this.materialFile=this.geometry.materials[0].mapDiffuse;
            }

        }
     }
    StageObject.prototype.createBuffers=function(gl){
        this.vbo = gl.createBuffer();
        this.ibo = gl.createBuffer();

        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, this.ibo);
        gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(this.geometry.indices),
gl.STATIC_DRAW);
        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, null);

        gl.bindBuffer(gl.ARRAY_BUFFER, this.vbo);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(this.geometry.vertices),
gl.STATIC_DRAW);
        this.vbo.itemSize = 3;
        this.vbo.numItems = this.geometry.vertices.length/3;

        this.nbo = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, this.nbo);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(this.geometry.normals),
gl.STATIC_DRAW);
        this.nbo.itemSize = 3;
        this.nbo.numItems = this.geometry.normals.length/3;

        if(this.materialFile!=null){

        this.verticesTextureBuffer = gl.createBuffer();
```

```
            gl.bindBuffer(gl.ARRAY_BUFFER, this.verticesTextureBuffer);
            gl.bufferData(gl.ARRAY_BUFFER, new
Float32Array(this.geometry.uvs[0]),gl.STATIC_DRAW);
            }

        }
    StageObject.prototype.clone=function(){
        var stageObject=new StageObject();
        stageObject.geometry=this.geometry.clone();

        var i;
        for(i=0;i<this.diffuseColor.length;++i){
            stageObject.diffuseColor[i]=this.diffuseColor[i];
        }
        for(i=0;i<this.ambientColor.length;++i){
            stageObject.ambientColor[i]=this.ambientColor[i];
        }
        for(i=0;i<this.specularColor.length;++i){
            stageObject.specularColor[i]=this.specularColor[i];
        }
        stageObject.rotationX=this.rotationX;
        stageObject.name=this.name;
        stageObject.rotationY=this.rotationY;
        stageObject.rotationZ=this.rotationZ;
        stageObject.materialFile=this.materialFile;
        stageObject.textureIndex=this.textureIndex;
        stageObject.loaded=true;
        return stageObject;

    }
    StageObject.prototype.update=function(){
        mat4.identity(this.modelMatrix);

        mat4.translate(this.modelMatrix,this.modelMatrix, this.location);

        mat4.rotateX(this.modelMatrix,this.modelMatrix,this.rotationX);
        mat4.rotateY(this.modelMatrix,this.modelMatrix,this.rotationY);
        mat4.rotateZ(this.modelMatrix,this.modelMatrix,this.rotationZ);

    }


    Light = function (  ) {

        this.specularColor=[0,0,0];
        this.diffuseColor=[0,0,0];
        this.ambientColor=[0,0,0];
        this.direction=[];
        this.position=[];
    };
    Light.prototype = {

        constructor: Light
    }


    Lights = function ( ) {
```

```
    this.lights=[];
};
Lights.prototype = {

    constructor: Lights,
    addLight:function(light){
        this.lights.push(light);
    },
    getDataByType:function(type,lightType){
        var sum=[];

        if(type=="position"){
         for(var i=0;i<this.lights.length;++i){
             if(this.lights[i].position.length>0){
                 sum=sum.concat(this.lights[i][type])
             }
         }

        } else if(type=="direction"){
            for(var i=0;i<this.lights.length;++i){
                if(this.lights[i].direction.length>0){
                    sum= sum.concat(this.lights[i][type])
                }
            }

        }
        else if(type=="ambientColor"){
            sum=[0,0,0];
            for(var i=0;i<this.lights.length;++i){

                    sum[0]=sum[0]+this.lights[i].ambientColor[0];
                sum[1]=sum[1]+this.lights[i].ambientColor[1];
                sum[2]=sum[2]+this.lights[i].ambientColor[2];


            }
        }
        else{
            //Since in our shader we expect color(diffuse and specular) of directional lights first then positional lights
            for(var i=0;i<this.lights.length;++i){
                if(this.lights[i][lightType].length>0){
                    sum= sum.concat(this.lights[i][type])
                }
            }
        }
        return sum;
    }
}
```

ДОДАТОК Б

Код вебгри

```
var Chef= function (){
   StageObject.apply(this);
   this.Keys = {
      up: false,
      down: false,
      left: false,
      right: false
   };
   this.document = null;
   this.item = null;
   this.picked = false;
   this.angle = Math.atan2(this.location[0], this.location[2]);
   this.direction = new CANNON.Vec3(0,0,1);
   this.tables;
   this.fromTable = false;
   this.toTable = false;
   this.table = null;
   this.plate=null;
};
Chef.prototype = Object.create(StageObject.prototype);
Chef.prototype.constructor = Chef;
Chef.prototype.initializePhysics=function(){
   const GROUP1 = 1
   const GROUP2 = 2
   const GROUP3 = 4
   const GROUP4 = 8
   var box = new CANNON.Body({
      mass: 150, // kg
      position: new CANNON.Vec3(0, 0, 0), // m
      shape: new CANNON.Box(new CANNON.Vec3(1, 1, 1)),
      collisionFilterGroup: GROUP2, // Put the sphere in group
      collisionFilterMask: GROUP2 | GROUP1
   });
   box.angularDamping = 0.9;
   this.rigidBody=box;
   this.system.addBody(box);


}

Chef.prototype.update=function() {
    if(this.rigidBody){
      var pos = this.rigidBody.position;
      mat4.identity(this.modelMatrix);
      this.location=vec3.fromValues(pos.x, pos.y, pos.z);
      mat4.translate(this.modelMatrix,this.modelMatrix, this.location);
      mat4.scale(this.modelMatrix,this.modelMatrix,vec3.fromValues(1.2,1.2,1.2));

    }  else{
```

```
        mat4.identity(this.modelMatrix);

        mat4.translate(this.modelMatrix,this.modelMatrix, this.location);

        mat4.rotateX(this.modelMatrix,this.modelMatrix,this.rotationX);
        mat4.rotateY(this.modelMatrix,this.modelMatrix,this.rotationY);
        mat4.rotateZ(this.modelMatrix,this.modelMatrix,this.rotationZ);
    }
    const GROUP1 = 1
    const GROUP2 = 2
    const GROUP3 = 4
    const GROUP4 = 8

    if(this.plate == null && this.tables){
        for(var t in this.tables){
            if(tables[t].food!=0){
                try{
                    if(tables[t].food.name == "plate"){
                    this.plate =tables[t].food;
                    break;
                    }
                }
                catch{}

            }
        }
    }
```

```
    const from = new CANNON.Vec3(this.rigidBody.position.x+this.direction.x*2, 1, this.rigidBody.position.z+this.direction.z*2);
    const to = new CANNON.Vec3(this.rigidBody.position.x+this.direction.x*5, 1, this.rigidBody.position.z+this.direction.z*5);
    var ray = new CANNON.Ray(from, to);
    ray.intersectWorld(this.system, {mode: CANNON.Ray.CLOSEST, collisionFilterGroup: GROUP3, collisionFilterMask: GROUP2 | GROUP4});

    if(ray.hasHit&&!this.picked){

        for(var t in this.tables){
            if(tables[t].rigidBody==ray.result.body && tables[t].food && tables[t].food.stack) {
                this.fromTable = true;
                this.item = tables[t].food.stack;
                this.table = null;
                break;
            }
            else if(tables[t].rigidBody==ray.result.body && tables[t].food!=0){
                this.fromTable = true;
                this.table = tables[t];
                if(this.plate && this.table.food == this.plate.rigidBody) this.item = this.plate;
                else this.item = this.table.food;
                break;
            }
            else if(tables[t].rigidBody==ray.result.body){
                this.fromTable = true;
                break;
            }
```

```
                else this.fromTable = false;
            }
            if(!this.fromTable){
                if(this.plate && ray.result.body == this.plate.rigidBody) this.item = this.plate;
                else this.item = ray.result.body;
            }
        }
        else if(!ray.hasHit&&!this.picked){
            this.item = null;
            this.table=null;
            this.fromTable = false;
            this.toTable = false;
        }
        else if(ray.hasHit&&this.picked){
            for(var t in this.tables){
                if(tables[t].rigidBody==ray.result.body) {
                    this.toTable = true;
                    this.table = tables[t];
                    break;
                }
            }
        }
        else if(!ray.hasHit&&this.picked){
            this.table=null;
            this.fromTable = false;
            this.toTable = false;
        }

        var Keys = this.Keys;
        var chef = this;

         this.document.onkeydown = function(e){
            var kc = e.keyCode;
            e.preventDefault();
            if(kc === 65) Keys.left = true;
            if(kc === 87) Keys.up = true;
            if(kc === 68) Keys.right = true;
            if(kc === 83) Keys.down = true;
            if(kc === 69) {
                if(!chef.picked&&chef.item!=null){
                    chef.picked = true;
                    //console.log(chef.item);
                    if(chef.fromTable&&chef.table==null){
                        //console.log(chef.table);
                        chef.item = chef.item.pop();
                        chef.item.place       =vec3.fromValues(chef.rigidBody.position.x+chef.direction.x*3,
chef.rigidBody.position.y, chef.rigidBody.position.z+chef.direction.z*3);
                        chef.item.location    =vec3.fromValues(chef.rigidBody.position.x+chef.direction.x*3,
chef.rigidBody.position.y, chef.rigidBody.position.z+chef.direction.z*3);
                        chef.item.initializePhysics();
                        chef.item.visible = true;
                        chef.item = chef.item.rigidBody;
                        chef.fromTable = false;
                    }
                    else if(chef.fromTable&&chef.table){
                        if(chef.item == chef.plate){
                            chef.plate.deleteConstraint();
```

```
                    //chef.item.rigidBody.collisionFilterMask = GROUP1 | GROUP2 | GROUP3 |
GROUP4;
                }
                else if(!chef.table.food.rigidBody) chef.item = chef.table.food;
                else chef.item = chef.table.food.rigidBody;
                chef.table.food = 0;
                chef.fromTable = false;
                chef.table = null;
            }
            chef.item.collisionFilterMask = GROUP1 | GROUP2 | GROUP3 | GROUP4;
        } else{
            if(chef.toTable && chef.table.name == "bin"){
                if(chef.item == chef.plate){
                    chef.table.items = chef.plate.food;
                    chef.plate.food = [];
                    chef.item.food = [];
                }
                else{
                    chef.table.items.push(chef.item);
                    chef.item == null;
                    chef.picked = false;
                }

            }
            else if(chef.toTable && chef.table.name == "order" && chef.item == chef.plate){

                chef.table.plate = chef.plate;
                chef.item.rigidBody.collisionFilterMask = GROUP4 | GROUP3 | GROUP2;
                chef.item.rigidBody.position = new CANNON.Vec3(chef.table.location[0]+1,
chef.table.location[1]+1, chef.table.location[2]) ;
                chef.item = null;
                chef.picked = false;
            }
            else if(chef.toTable&&chef.table.food==0){
                if(chef.item == chef.plate){
                    chef.item.addConstraint(chef.table);
                    chef.item = chef.item.rigidBody;
                }

                chef.item.position = new CANNON.Vec3(chef.table.location[0],
chef.table.location[1]+1, chef.table.location[2]) ;
                chef.item.collisionFilterMask = GROUP4 | GROUP3 | GROUP2 ;
                chef.table.food = chef.item;
                chef.item = null;
                chef.picked = false;
            }
            else if(chef.toTable&&(chef.table.food==chef.plate.rigidBody ||
chef.table.food.rigidBody==chef.plate.rigidBody)&&chef.plate.food.length<3){
                chef.item.collisionFilterMask = GROUP4 | GROUP3 | GROUP2;
                chef.item.position = new CANNON.Vec3(chef.plate.location[0],
chef.plate.location[1]+1, chef.plate.location[2]) ;
                chef.plate.food.push(chef.item);

                chef.item = null;
                chef.picked = false;
            }
            else{
```

```
                chef.item = null;
                chef.picked = false;
            }
            chef.table = null;
            chef.toTable = false;
        }
    }
};

if(this.picked&&this.item!=null){

    if(this.item == this.plate){
        this.plate.direction = new CANNON.Vec3(this.direction.x, this.direction.y,
this.direction.z)
        this.item.rotationY = this.rotationY;
        this.item.rigidBody.position = new
CANNON.Vec3(this.rigidBody.position.x+this.direction.x*3, this.rigidBody.position.y+2,
this.rigidBody.position.z+this.direction.z*3);
    }
        else this.item.position = new CANNON.Vec3(this.rigidBody.position.x+this.direction.x*3,
this.rigidBody.position.y, this.rigidBody.position.z+this.direction.z*3);
    }
    if(this.plate&&this.plate.food.length>0){
        if(this.plate.direction.x!=0&&this.plate.direction.z!=0){
            if(this.plate.food[0]) this.plate.food[0].position = new
CANNON.Vec3(this.plate.location[0]-this.plate.direction.z*1.2, this.plate.location[1]+0.2,
this.plate.location[2]+this.plate.direction.x*0.8) ;
            if(this.plate.food[1]) this.plate.food[1].position = new
CANNON.Vec3(this.plate.location[0], this.plate.location[1]+0.2, this.plate.location[2]) ;
            if(this.plate.food[2]) this.plate.food[2].position = new
CANNON.Vec3(this.plate.location[0]+this.plate.direction.z*0.8, this.plate.location[1]+0.2,
this.plate.location[2]-this.plate.direction.x*1.2) ;
        }
        else{
            if(this.plate.food[0]) this.plate.food[0].position = new
CANNON.Vec3(this.plate.location[0]+this.plate.direction.z, this.plate.location[1]+0.2,
this.plate.location[2]+this.plate.direction.x) ;
            if(this.plate.food[1]) this.plate.food[1].position = new
CANNON.Vec3(this.plate.location[0], this.plate.location[1]+0.2, this.plate.location[2]) ;
            if(this.plate.food[2]) this.plate.food[2].position = new
CANNON.Vec3(this.plate.location[0]-this.plate.direction.z, this.plate.location[1]+0.2,
this.plate.location[2]-this.plate.direction.x) ;

        }
        }

this.document.onkeyup = function(e){
    var kc = e.keyCode;
    e.preventDefault();

    if(kc === 65) Keys.left = false;
    if(kc === 87) Keys.up = false;
    if(kc === 68) Keys.right = false;
    if(kc === 83) Keys.down = false;
};

this.Keys = Keys;
```

```
    if(this.Keys){
      var pos = this.rigidBody.position;
      if(this.Keys.up){
         this.rigidBody.position.z-=1;
         this.direction.z=-1;
         if(!this.Keys.left&&!this.Keys.right) this.direction.x=0;
      }

      if(this.Keys.down){
         this.rigidBody.position.z+=1;
         this.direction.z=1;
         if(!this.Keys.left&&!this.Keys.right) this.direction.x=0;
      }

      if(this.Keys.left) {
         this.rigidBody.position.x-=1;
         this.direction.x=-1;
         if(!this.Keys.up&&!this.Keys.down) this.direction.z=0;
      }

      if(this.Keys.right){
         this.rigidBody.position.x+=1;
         this.direction.x=1;
         if(!this.Keys.up&&!this.Keys.down) this.direction.z=0;
      }

      var ax, az, tx, tz;
      ax = this.location[0];
      az = this.location[2];
      tx = this.rigidBody.position.x;
      tz = this.rigidBody.position.z;

      const desiredAngle = Math.atan2(tx-ax, tz-az);
      if(Object.values(this.Keys).includes(true)){
         this.rotationY=desiredAngle;
         this.angle = this.rotationY;
      }
      mat4.rotateY(this.modelMatrix,this.modelMatrix,this.rotationY);
      this.rigidBody.quaternion.setFromEuler(0, this.angle, 0);

   }
}


var Floor= function (){
   StageObject.apply(this);
};
Floor.prototype = Object.create(StageObject.prototype);
Floor.prototype.constructor = Floor;
Floor.prototype.initializePhysics=function(){

   const GROUP1 = 1
   const GROUP2 = 2
   const GROUP3 = 4
   const GROUP4 = 8
```

```
    var groundBody = new CANNON.Body({
        mass: 0,
        position: new CANNON.Vec3(0, 0, 0),
        collisionFilterGroup: GROUP2,
        collisionFilterMask: GROUP2
    });
    var groundShape = new CANNON.Plane();
    groundBody.addShape(groundShape);
    groundBody.quaternion.setFromEuler(-Math.PI / 2, 0, 0);
    this.rigidBody=groundBody;
    this.system.addBody(groundBody);

    this.system.addBody(createWall(new CANNON.Vec3(15,0, 0), 0, -Math.PI/2, 0));
    this.system.addBody(createWall(new CANNON.Vec3(-15,0, 0), 0, Math.PI/2, 0));
    this.system.addBody(createWall(new CANNON.Vec3(0,0, 15), 0, -Math.PI, 0));
    this.system.addBody(createWall(new CANNON.Vec3(0,0, -15), 0, 0, 0));
}

function createWall(position, rotX, rotY, rotZ){
    const GROUP1 = 1
    const GROUP2 = 2
    const GROUP3 = 4
    const GROUP4 = 8

    var groundBody = new CANNON.Body({
        mass: 0,
        position: position,
        collisionFilterGroup: GROUP1,
        collisionFilterMask: GROUP2
    });
    var groundShape = new CANNON.Plane();
    groundBody.addShape(groundShape);
    groundBody.quaternion.setFromEuler( rotX,rotY, rotZ);

    return groundBody;
}


var Food= function (){
    StageObject.apply(this);
    this.stack = [];
    this.place = null;
};
Food.prototype = Object.create(StageObject.prototype);
Food.prototype.constructor = Food;
Food.prototype.initializePhysics=function(){
    const GROUP1 = 1
    const GROUP2 = 2
    const GROUP3 = 4
    const GROUP4 = 8

    const halfExtents = new CANNON.Vec3(1, 1, 1)
    const boxShape = new CANNON.Box(halfExtents)
    var boxBody = new CANNON.Body({
        mass: 1, // kg
        position: new CANNON.Vec3(this.place[0], this.place[1], this.place[2]), // m
        shape: boxShape,
```

```
        collisionFilterGroup: GROUP2,
        collisionFilterMask: GROUP4
    });

    this.rigidBody=boxBody;
    this.system.addBody(boxBody);
}
Food.prototype.clone=function(){
    var stageObject=new Food();
    stageObject.geometry=this.geometry.clone();
    stageObject.system = this.system;
    var i;
    for(i=0;i<this.diffuseColor.length;++i){
        stageObject.diffuseColor[i]=this.diffuseColor[i];
    }
    for(i=0;i<this.ambientColor.length;++i){
        stageObject.ambientColor[i]=this.ambientColor[i];
    }
    for(i=0;i<this.specularColor.length;++i){
        stageObject.specularColor[i]=this.specularColor[i];
    }
    stageObject.rotationX=this.rotationX;
    stageObject.name=this.name;
    stageObject.rotationY=this.rotationY;
    stageObject.rotationZ=this.rotationZ;
    stageObject.materialFile=this.materialFile;
    stageObject.textureIndex=this.textureIndex;
    stageObject.loaded=true;
    return stageObject;

}
Food.prototype.update=function(){
    if(this.rigidBody){
        var pos = this.rigidBody.position;
        mat4.identity(this.modelMatrix);
        this.location=vec3.fromValues(pos.x, pos.y, pos.z);
        mat4.translate(this.modelMatrix,this.modelMatrix, this.location);

        mat4.scale(this.modelMatrix,this.modelMatrix,vec3.fromValues(1.2,1.2,1.2));

     }   else{
         mat4.identity(this.modelMatrix);

         mat4.translate(this.modelMatrix,this.modelMatrix, this.location);

         mat4.rotateX(this.modelMatrix,this.modelMatrix,this.rotationX);
         mat4.rotateY(this.modelMatrix,this.modelMatrix,this.rotationY);
         mat4.rotateZ(this.modelMatrix,this.modelMatrix,this.rotationZ);
         //mat4.scale(this.modelMatrix,this.modelMatrix,vec3.fromValues(10,10,10));
     }
}


var Kitchen= function (){
    StageObject.apply(this);
    this.food=0;
};
```

```
Kitchen.prototype = Object.create(StageObject.prototype);
Kitchen.prototype.constructor = Kitchen;
Kitchen.prototype.initializePhysics=function(){
   const GROUP1 = 1
   const GROUP2 = 2
   const GROUP3 = 4
   const GROUP4 = 8

   const halfExtents = new CANNON.Vec3(4,2, 4)
   const boxShape = new CANNON.Box(halfExtents)
   var boxBody = new CANNON.Body({
      mass: 0, // kg
      position: new CANNON.Vec3(this.location[0], this.location[1], this.location[2]), // m
      shape: boxShape,
      collisionFilterGroup: GROUP4, // Put the sphere in group 1
      collisionFilterMask: GROUP2 | GROUP3
   });
   this.rigidBody=boxBody;
   this.system.addBody(boxBody);
}
Kitchen.prototype.clone=function(){
   var stageObject=new Kitchen();
   stageObject.geometry=this.geometry.clone();

   var i;
   for(i=0;i<this.diffuseColor.length;++i){
      stageObject.diffuseColor[i]=this.diffuseColor[i];
   }
   for(i=0;i<this.ambientColor.length;++i){
      stageObject.ambientColor[i]=this.ambientColor[i];
   }
   for(i=0;i<this.specularColor.length;++i){
      stageObject.specularColor[i]=this.specularColor[i];
   }
   stageObject.rotationX=this.rotationX;
   stageObject.name=this.name;
   stageObject.rotationY=this.rotationY;
   stageObject.rotationZ=this.rotationZ;
   stageObject.materialFile=this.materialFile;
   stageObject.textureIndex=this.textureIndex;
   stageObject.loaded=true;
   return stageObject;

}
Kitchen.prototype.update=function(){
   if(this.rigidBody){
      var pos = this.rigidBody.position;
      mat4.identity(this.modelMatrix);
      this.location=vec3.fromValues(pos.x, pos.y, pos.z);
      mat4.translate(this.modelMatrix,this.modelMatrix, this.location);

      //mat4.scale(this.modelMatrix,this.modelMatrix,vec3.fromValues(5,5,5));

    }  else{
       mat4.identity(this.modelMatrix);

       mat4.translate(this.modelMatrix,this.modelMatrix, this.location);
```

```
        mat4.rotateX(this.modelMatrix,this.modelMatrix,this.rotationX);
        mat4.rotateY(this.modelMatrix,this.modelMatrix,this.rotationY);
        mat4.rotateZ(this.modelMatrix,this.modelMatrix,this.rotationZ);
        //mat4.scale(this.modelMatrix,this.modelMatrix,vec3.fromValues(10,10,10));
    }
}


var Order= function (){
   StageObject.apply(this);
   this.stage;
   this.order = [];
   this.plate;
   this.tables;
   this.boards = [];
   this.food_names = [];
   this.clear_texture;
};
Order.prototype = Object.create(StageObject.prototype);
Order.prototype.constructor = Order;
Order.prototype.initializePhysics=function(){
   const GROUP1 = 1
   const GROUP2 = 2
   const GROUP3 = 4
   const GROUP4 = 8

   const halfExtents = new CANNON.Vec3(3,4, 4)
   const boxShape = new CANNON.Box(halfExtents)
   var boxBody = new CANNON.Body({
      mass: 0, // kg
      position: new CANNON.Vec3(this.location[0], this.location[1], this.location[2]-1), // m
      shape: boxShape,
      collisionFilterGroup: GROUP4,
      collisionFilterMask: GROUP2 | GROUP3
   });
   this.rigidBody=boxBody;
   this.system.addBody(boxBody);
}
Order.prototype.clone=function(){
   var stageObject=new Order();
   stageObject.geometry=this.geometry.clone();

   var i;
   for(i=0;i<this.diffuseColor.length;++i){
      stageObject.diffuseColor[i]=this.diffuseColor[i];
   }
   for(i=0;i<this.ambientColor.length;++i){
      stageObject.ambientColor[i]=this.ambientColor[i];
   }
   for(i=0;i<this.specularColor.length;++i){
      stageObject.specularColor[i]=this.specularColor[i];
   }
   stageObject.rotationX=this.rotationX;
   stageObject.name=this.name;
   stageObject.rotationY=this.rotationY;
   stageObject.rotationZ=this.rotationZ;
```

```
        stageObject.materialFile=this.materialFile;
        stageObject.textureIndex=this.textureIndex;
        stageObject.loaded=true;
        return stageObject;

    }
    Order.prototype.update=function(){
        if(this.rigidBody){
            var pos = this.rigidBody.position;
            mat4.identity(this.modelMatrix);
            this.location=vec3.fromValues(pos.x, pos.y, pos.z);
            mat4.translate(this.modelMatrix,this.modelMatrix, this.location);
        }  else{
            mat4.identity(this.modelMatrix);

            mat4.translate(this.modelMatrix,this.modelMatrix, this.location);

            mat4.rotateX(this.modelMatrix,this.modelMatrix,this.rotationX);
            mat4.rotateY(this.modelMatrix,this.modelMatrix,this.rotationY);
            mat4.rotateZ(this.modelMatrix,this.modelMatrix,this.rotationZ);
        }

        if(!this.clear_texture && this.boards.length>0) this.clear_texture = this.boards[0].textureIndex;
        if(this.boards.length>0 && this.order.length == 0) MakeOrder(this);

        if(this.plate){
            var order=[];
            for(var i=0; i<this.order.length; i++){
                order.push(this.order[i]);
            }
            if(this.plate.food.length > 0){
                for(var i=0; i<this.plate.food.length; i++){
                    for(var o in this.stage.stageObjects){
                        if(this.stage.stageObjects[o].rigidBody == this.plate.food[i]){

                            if(order.includes(this.stage.stageObjects[o].name)){
                                const index =  order.indexOf(this.stage.stageObjects[o].name);
                                order.splice(index, 1);
                            }
                            const index =  this.stage.stageObjects.indexOf(this.stage.stageObjects[o]);
                            this.stage.stageObjects.splice(index, 1);
                            break;
                        }
                    }
                }
                if(this.order.length == this.plate.food.length && order.length == 0) this.order = [];
                this.plate.food = [];
            }

            var num =25;
            tables[num].food = this.plate;
            this.plate.addConstraint(tables[num]);
            this.plate.rigidBody.position = new CANNON.Vec3(tables[num].location[0],
tables[num].location[1]+0.1, tables[num].location[2]);
            this.plate.rotationY = -Math.PI/2;
            this.plate.rigidBody.linearVelocity = 0;
            this.plate = null;
```

```
    }
}

function MakeOrder(obj){
   var count = Math.floor(Math.random() * 3) + 1;

   for(var i=0; i<count; i++){
      var item = Math.floor(Math.random() * obj.food_names.length);
      obj.order.push(obj.food_names[item]);
   }

   for(var i=0; i<3; i++){
      if(obj.order[i]){
         for(var o in obj.stage.stageObjects){
            if(obj.stage.stageObjects[o].name == obj.order[i]){
               obj.boards[i].textureIndex = obj.stage.stageObjects[o].textureIndex;
            }
         }
      }
      else obj.boards[i].textureIndex = obj.clear_texture;

   }
}


var Plate= function (){
   StageObject.apply(this);
   this.food = [];
   this.place = null;
   this.direction = new CANNON.Vec3(1,0,0);
   this.constraint;
};
Plate.prototype = Object.create(StageObject.prototype);
Plate.prototype.constructor = Plate;
Plate.prototype.initializePhysics=function(){
   const GROUP1 = 1
   const GROUP2 = 2
   const GROUP3 = 4
   const GROUP4 = 8

   const halfExtents = new CANNON.Vec3(1, 1, 1);
   const boxShape = new CANNON.Box(halfExtents)
   var boxBody = new CANNON.Body({
      mass: 1, // kg
      position: new CANNON.Vec3(this.place[0], this.place[1], this.place[2]), // m
      shape: boxShape,
      collisionFilterGroup: GROUP2, // Put the sphere in group 1
      collisionFilterMask:  GROUP4 | GROUP2
   });
   boxBody.linearDamping = 0.95;
   this.rigidBody=boxBody;
   this.system.addBody(boxBody);
}
Plate.prototype.clone=function(){
   var stageObject=new Plate();
   stageObject.geometry=this.geometry.clone();
```

```
    stageObject.system = this.system;
    var i;
    for(i=0;i<this.diffuseColor.length;++i){
        stageObject.diffuseColor[i]=this.diffuseColor[i];
    }
    for(i=0;i<this.ambientColor.length;++i){
        stageObject.ambientColor[i]=this.ambientColor[i];
    }
    for(i=0;i<this.specularColor.length;++i){
        stageObject.specularColor[i]=this.specularColor[i];
    }
    stageObject.rotationX=this.rotationX;
    stageObject.name=this.name;
    stageObject.rotationY=this.rotationY;
    stageObject.rotationZ=this.rotationZ;
    stageObject.materialFile=this.materialFile;
    stageObject.textureIndex=this.textureIndex;
    stageObject.loaded=true;
    return stageObject;
}
Plate.prototype.update=function(){
    if(this.rigidBody){
        var pos = this.rigidBody.position;
        mat4.identity(this.modelMatrix);
        this.location=vec3.fromValues(pos.x, pos.y, pos.z);
        mat4.translate(this.modelMatrix,this.modelMatrix, this.location);

        mat4.scale(this.modelMatrix,this.modelMatrix,vec3.fromValues(1.5,1.5,1.5));
        this.rigidBody.quaternion.y = this.rotationY;
        mat4.rotateY(this.modelMatrix,this.modelMatrix,this.rotationY);

     }   else{
         mat4.identity(this.modelMatrix);

         mat4.translate(this.modelMatrix,this.modelMatrix, this.location);

         mat4.rotateX(this.modelMatrix,this.modelMatrix,this.rotationX);
         mat4.rotateY(this.modelMatrix,this.modelMatrix,this.rotationY);
         mat4.rotateZ(this.modelMatrix,this.modelMatrix,this.rotationZ);
         //mat4.scale(this.modelMatrix,this.modelMatrix,vec3.fromValues(10,10,10));
     }

}
Plate.prototype.addConstraint=function(table){
    this.constraint = new CANNON.PointToPointConstraint(
        this.rigidBody,
        new CANNON.Vec3(0, 0, 0),
        table.rigidBody,
        new CANNON.Vec3(0, 2.5, 0)
      )
      this.system.addConstraint(this.constraint);
}
Plate.prototype.deleteConstraint=function(){
    this.system.removeConstraint(this.constraint);
    this.constraint = null;
}
```

```
var Bin= function (){
   StageObject.apply(this);
   //this.visible=true;
   this.items = [];
   this.stage;
};
Bin.prototype = Object.create(StageObject.prototype);
Bin.prototype.constructor = Bin;
Bin.prototype.initializePhysics=function(){
   const GROUP1 = 1
   const GROUP2 = 2
   const GROUP3 = 4
   const GROUP4 = 8

   const halfExtents = new CANNON.Vec3(4,4,4)
   const boxShape = new CANNON.Box(halfExtents)
   var boxBody = new CANNON.Body({
      mass: 0, // kg
      position: new CANNON.Vec3(this.location[0], this.location[1], this.location[2]), // m
      shape: boxShape,
      collisionFilterGroup: GROUP4, // Put the sphere in group 1
      collisionFilterMask: GROUP2 | GROUP3
   });
   this.rigidBody=boxBody;
   this.system.addBody(boxBody);
}
Bin.prototype.clone=function(){
   var stageObject=new Bin();
   stageObject.geometry=this.geometry.clone();

   var i;
   for(i=0;i<this.diffuseColor.length;++i){
      stageObject.diffuseColor[i]=this.diffuseColor[i];
   }
   for(i=0;i<this.ambientColor.length;++i){
      stageObject.ambientColor[i]=this.ambientColor[i];
   }
   for(i=0;i<this.specularColor.length;++i){
      stageObject.specularColor[i]=this.specularColor[i];
   }
   stageObject.rotationX=this.rotationX;
   stageObject.name=this.name;
   stageObject.rotationY=this.rotationY;
   stageObject.rotationZ=this.rotationZ;
   stageObject.materialFile=this.materialFile;
   stageObject.textureIndex=this.textureIndex;
   stageObject.loaded=true;
   return stageObject;

}
Bin.prototype.update=function(){
   if(this.rigidBody){
      var pos = this.rigidBody.position;
      mat4.identity(this.modelMatrix);
      this.location=vec3.fromValues(pos.x, pos.y, pos.z);
      mat4.translate(this.modelMatrix,this.modelMatrix, this.location);
```

```
        mat4.scale(this.modelMatrix,this.modelMatrix,vec3.fromValues(2.5,2.5,2.5));

    }   else{
        mat4.identity(this.modelMatrix);

        mat4.translate(this.modelMatrix,this.modelMatrix, this.location);

        mat4.rotateX(this.modelMatrix,this.modelMatrix,this.rotationX);
        mat4.rotateY(this.modelMatrix,this.modelMatrix,this.rotationY);
        mat4.rotateZ(this.modelMatrix,this.modelMatrix,this.rotationZ);
        //mat4.scale(this.modelMatrix,this.modelMatrix,vec3.fromValues(10,10,10));
    }

    if(this.items){
      if(this.items.length > 0){
          for(var i=0; i<this.items.length; i++){
              for(var o in this.stage.stageObjects){
                  if(this.stage.stageObjects[o].rigidBody == this.items[i]){

                      const index =  this.stage.stageObjects.indexOf(this.stage.stageObjects[o]);
                      this.stage.stageObjects.splice(index, 1);
                      delete this.items[i];
                      break;
                  }
              }

          }
          this.items = [];
      }
    }
}
```