Софийски университет „Св. Климент Охридски“

Факултет по математика и информатика

**ПРОЕКТ**

по учебна дисциплина

Мобилно Интернет Съдържание

**„Dice Simulator”**

***Изготвил:*** ***Ръководител:***

Георги Гергинов, 81966 проф. д-р Васил Цунижев

Компютърни науки, II курс

**Съдържание**

**Увод** ……………………………………………………………...... 3

Преглед на използваните източници в хода на разработката … 3

Преглед на използваните технологии в хода на разработката .. 4

**Проектиране** …………………………………………………......4

Описание режима на работа на симулатора ...…………………. 4

Технически особености…………………..………………….….. 5

Библиотеки и включени файлове …………………………………. 5

Функционалност на customScene.js ……………….………………. 6

Функционалност на index.html ……....………….………………. 11

**Тестване** ………………………………………...………………. 12

Извод от тестването ……………………………………………. 14

**Източници** ……………………………………………………… 14

# **Увод**

**„Dice simulator“** представлява триизмерен сензорно-контролиран симулатор на зарове за мобилни устройства. Целта е да бъде предоставен бърз, леснодостъпен и щадящ устройството виртуален заместител на истински зарове. Посредством улавянето на движенията на потребителското устройство и проектираните физически свойства на обектите, екранът на телефонът се превръща в игрално поле.

## **Преглед на използваните източници в хода на разработката**

**[1]** Документация на **threejs**

<https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene>

**[2]** Документация на **physijs**

<https://github.com/chandlerprall/Physijs/wiki/Basic-Setup>

**[3]** Справочник със събития

<https://developer.mozilla.org/en-US/docs/Web/Events>

**[4]** Hosting услуга

<https://www.netlify.com/>

**[5]** Дебъгване

https://developer.chrome.com/docs/devtools

В хода на създаването на проекта основно съм се опирал на **[1]** и **[2]**. В споменатите документации са разгледани както различните аспекти и функционалности на съответните библиотеки, така и прости примери илюстриращи как работят техните елементи помежду си, тъй като зависят една от друга. **[3]** спомогна с включването и изпълнението на събития, като отново, както и при гореспоменатите – са предоставени кратки примери. **[4]** е услугата, използвана за хостване на проекта в интернет и съответно тестване на различни физически устройства, а с помощта на насоките в **[5]** извърших успешно дебъгване на персоналния си компютър, когато беше нужно, както и тестване.

## **Преглед на използваните технологии в хода на разработката**

* **HTML5 –** Използва се за „скелета“ на уеб страницата, както и за включване на сензорни събития
* **JavaScript** – С **JS** е програмирана същината на проектната логика – сцена, обекти, взаимодействия и т.н.

# **Проектиране**

## **Описание режима на работа на симулатора**

Симулаторът е конструиран със замисъл да бъде използван единствено от мобилни устройства, тъй като само при тях може да бъде утилизиран наборът от пространствени сензори.

При стартиране, мобилното устройство трябва да бъде в портретен режим и вертикално положение. Изчакват се да паднат заровете върху игралното поле и да се регистрира първата колизия, след което всяко движение на телефона ще бъде отчетено и съответните физически сили ще бъдат приложени върху заровете. Поддържат се ротация и засилка на телефона във всички посоки, а гравитацията се смята вътрешно за библиотеките.

Контейнерът държащ заровете е проектиран така, че екранът на телефона да се възприема от потребителя като една от стените му, а самите му размери се смятат динамично за всяко устройство. Движенията са проектирани със замисъл да бъдат хиперболизирани, с цел да се постигат желаните ефекти с по-леки движения на устройството. Престоят в симулатора не е ограничен от събитие за край, т.е. е арбитрарен - в ръцете на потребителя.

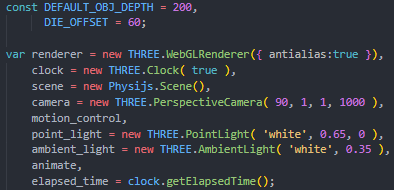
## **Технически особености**

### **Библиотеки и включени файлове**

* **threejs** – Това е библиотека за компютърна графика с JS и основата на този проект. Тя служи за съдаването и настройването на всичко в сцената – обекти, камери, светлини и т.н. Тъй като проекта не се нуждае от пълната функционалност на библиотеката е използвана **three.min.js** която съдържа само „най-важните“ неща от нея.
* **physijs** – Physijs надгражда над основата на threejs, като с нейна помощ се създават всички физически характеристики на обектите от threejs, както и прилагане на външни сили върху тези обекти. Как самите обекти реагират на тези сили се пресмята вътрешно за библиотеката, като за тази цел в работната директория трябва да бъдат прибавени **ammo.js** и **physijs\_worker.js**, които са част от physijs пакета. Накратко, ammo.js е директен порт на **Bullet physics engine** към JavaScript, а physijs е плъгин за threejs, който „интерпретира“ ammo.js, така че да бъде приложим към обектите от threejs. physijs\_worker.js отговаря за изпълнение на симулацията на сцената върху друга от основната нишка за да се избегне претоварване и загуба на качество, но за сметка на това се стига до разминавания на нишките.
* **WebGL.js** – Това представлява файл съдържащ откъс код, който проверява дали използваните от потребителя устройство и браузер поддържат WebGL. Проверката се извършва, защото WebGL е императивно за работата на threejs. Това е функционалност, която се съдържа в threejs, но тъй като използваме „min“ версията, както вече споменах, се налага да прибавим този скрипт отделно.
* **DeviceOrientationControls.js** – Подобно на WebGL.js и това е съществуваща функционалност, която трябва допълнително да добваим. Както подсказва името на файла, този скрипт се грижи за ротацията на мобилното устройство. Извършва се посредством събития за промяна в ориентацията на устройството и вградените в threejs функционалности грижещи се за манипулацията на ротационни тела.
* **customScene.js** – Тук се намира по-голямата част от логиката на самия проект, като е изведена в този файл и разделено на функции с цел по-лесна четимост и модуларност.

### **Функционалност на customScene.js**

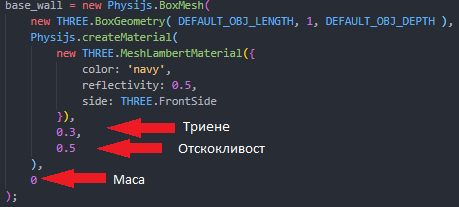
В началото на файла са изведени всички променливи и константи като глобални, за да могат да бъдат използвани както в този файл, така и в скрипт секцията на уеб страницата, след като той бъде прибавен. Избрано е да бъде така, защото манипулация на код има и в гореспоменатата скрипт секция, за което се изисква декларацията на променливите да не е локална, при положение, че са в друг файл.



Фигура 1. Глобални декларации

Функциите следва да бъдат разгледани в азбучен ред.

* **containerInit()** – Тази функция отговаря за създаването, позиционирането и определянето на физическия профил на контейнера. Той представлява сложен обект създаден от 6 стени и 8 подпори и невидим обект действащ като център на фигурата. Една от стените съвпада с екрана на устройството и е направена невидима, с цел да бъде създадена илюзията, че телефонът е самия контейнер. Има дълбочина **DEFAULT\_OBJ\_DEPTH** (вж. Фиг. 1) и както вече бе споменато, ширината и височината на контейнера съвпадат с тези на мобилния екран. Един от преобладаващите проблеми е, че движението на устройството може да бъде твърде рязко за симулацията и по този начин да бъде изпусната колизия между стена и зар. Допълнителните подпори и физическите характеристики на контейнера се борят именно с това. Важно е да се отбележи, че към невидимия център са закачени камерата и светлината – тъй като около него ще се извършва ротацията на самия контейнер и закрепявайки ги осигуряваме, че осветлението ще бъде изправно и гледната точка няма да се променя при завъртане. Целта е при завъртане на телефона да се върти контейнера, а заровете да „реагират“ на физическите сили, които им се прилагат.



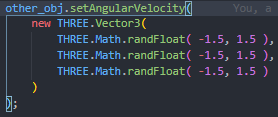
Фигура 2.1 Създаване на стена



Фигура 2.2 Добавяне на събитие за регистриране на сблъсък

* **collision()** – Тази функция се грижи за промените върху обектите при засечена колизия. Проектирано е така, че при удар, освен натуралната реакция, която зарът би имал, да се прибави и съвсем леко побутване обратно, към сърцевината на контейнера. Посоката на вектора се смята с линейна интерполация между позицията на зара и тази на центъра. Това побутване държи заровете подвижни и ги пази от преминаване на установените граници. Като допълнение, се прилага и произволен въртящ момент на зарчето, отново с цел да има усет за произволност на резултата и леснота на движението.

Фигура 3 Настройване на въртящ момент

****

* **diceInit()** – С diceInit() създаваме, позиционираме и настройваме заровете. Всеки от тях е сложен обект съставен от 3 паралелепипеда, 8 сфери, 12 цилиндъра и 21 кръга. Построяват се аналогично на контейнера, само че им се дава случайна стартова позиция в границите на контейнера и случайно стартово завъртане, които при всяко зареждане на страницата ще бъдат различни. Именно защото стартовите им позиции са във въздуха и отнема секунда да реагират на гравитацията, телефонът трябва да седи изправен до падането им. При наклон на устройството, позицията на контейнера в пространството ще се промени, но използвания софтуер не позволява заровете да се преместят на нова позиция без да се наруши симулацията. Макар и заровете да са идентични, не можем просто да ги клонираме, тъй като в такъв случай ще загубим възможността да настроим физическия профил на отделните елементи на клонинга. За разлика от контейнера, заровете имат допълнителна настройка за бързината на затихване на инерцията им във въздуха.
* **deviceMotion()** – deviceMotion() се грижи за отчитане на стойностите предоставени от акселерометъра на телефона и прилагането им върху заровете. Изработено е така, че заровете да реагират спрямо Трети закон на Нютон – с еднакво по големина и обратно по сила противодействие, като спазва и ограниченията на контейнера.



Фигура 4. Прибавяне на събитие за акселерометъра

* **frame()** – Тази функция служи за помощ със симулацията в скрипт секцията на уеб страницата и рендерването на картината. Служи за обновяването на визуалната репрезентация от пускането на страницата нататък.
* **outOfBoundsAnimationHandler()** – В себе си съдържа редица условия, които контролират как заровете реагират при достигане на границите на контейнера и действа като „помощна“ на функцията отчитаща колизии. Разликата е, че тези условия, за разлика от събитието за сблъсъци се проверяват активно всеки фрейм, в цикъла отговарящ за симулацията на картината.
* **onWindowResize()** – Пояснява държанието на рендерера на картината и камерата при събитието за промяна на размера на прозореца и инструктира генерираната картина да запълва уеб страницата.
* **sceneInit()** – Основните обекти от сцената се инициализират още в глобалните променливи (вж. Фиг. 1), но някои от тях се нуждаят от допълнителни настройки, които се извършват именно в sceneInit()

****

Фигура 5. Тялото на sceneInit()

* **visibleHeight()** – Пресмята и връща на какво разстояние в threejs мерни единици отговаря височината на екрана на устройството в зависимост от това колко отдалечена камерата в програмата и какво е нейното поле на видимост.
* **visibleWidth()** - Аналогична на горната, само че за широчината на екрана.

### **Функционалност на index.html**

Съдържанието на **index.html** е сведено до възможно най-семпло, така че ще обърнем внимание само на кода в <script> тага.

Освен, че се викат разгледаните по-горе Init() функции инициализиращи съответните обекти и се правят малки позиционни корекции в камерата и светлината, тук се инициализира и контролът над жироскопа на устройството. Използва се класът **DeviceOrientationControls** с център на ротацията геометричния център на контейнера, който е невидимият обект описан в containerInit(). Както обосновахме в секция **2.2.1**, обработването на ротационното събитие се случва в DeviceOrientationControls.js.

Непосредствено под тази инициализация се извършва и обновяването на данните за завъртането на телефона, проверките от **outOfBoundsAnimationHandler()** и самата симулация на физиките в сцената, като всяка стъпка от нея е настроена да се извършва за 1 милисекунда.

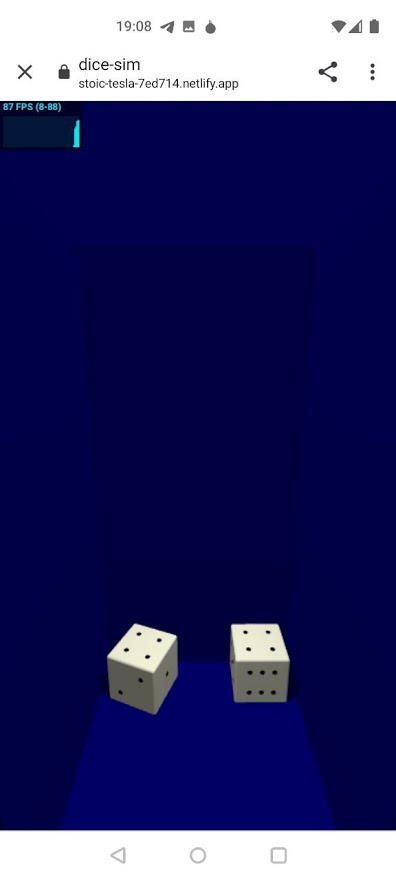
Всичко това се случва в animate() функцията, която е част от threejs и контролира събитията в сцената за всеки фрейм от пускането. Важно е да се отбележи, че данните за позицията на контейнера във физическата симулация се опресняват на всеки фрейм с помощта на dirtyRotation атрибутите на компонентите на контейнера – това трябва да се случва в animate(), защото в противен случай опресняването би било еднократно.

# **Тестване**

За да бъде тествано приложението, уеб страницата беше хостната в интернет с помощта на безплатните услуги на **Netlify**.

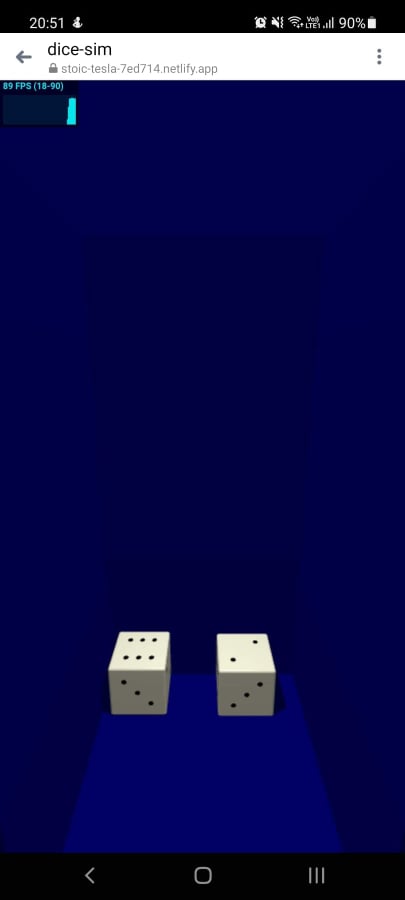
Самия процес на тестване се състоя от ръчно изпробване върху телефони от различни производители (Google, Apple, OnePlus, Samsung) на набор от няколко браузера (Chrome, Safari, DuckDuckGo).

За повече контекст относно представянето на симулатора върху различните устройства единствено за тестовете ще бъде добавен модула **stats.min.js**, който изписва данни на живо в горния ляв ъгъл на прозореца. Той **няма** да бъде част от крайния продукт. Ще се фокусираме върху фреймовете в секунда (**FPS**), тъй като за потребителското изживяване е най-важно, а и рефлектира натовареността на самите устройства.



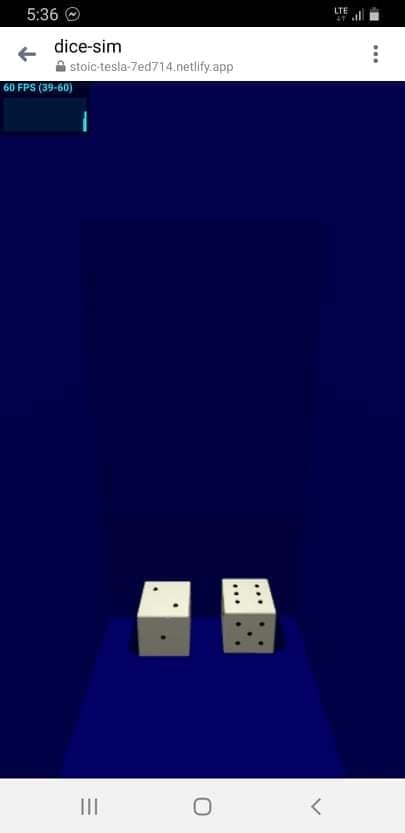
Фигура 6.1 Google Pixel 2

Фигура 6.2 OnePlus Nord



Фигура 6.4 Samsung S9

Фигура 6.3 OnePlus 5T



Фигура 6.5 Samsung S8

Фигура 6.6 OnePlus 8T

## **Извод от тестването**

Изводът от тестването е, че представянето на различните телефони почти изцяло зависи от хардуера на устройството. Всички от изброените водещи браузъри поддържат WebGL и безпроблемно подкарват скриптовете на симулатора.

Основните фактори за представянето са 2 – честотата на опресняване на екрана на съответното устройство и хардуера му. От първото зависи каква ще бъде горната граница на фреймите в секунда, съответно колко плавна ще бъде картината, а от второто дали ще има падежи във фреймите и колко бързо устройството ще се възстанови. При първата колизия се забелязва спад в представянето при почти всички, като при по-старите технологии е по-силно изявено, но възстановяването до максимални фрейми в секунда никъде не отнема повече от секунда. С изключение на този първоначален скок, симулацията протича безпроблемно.

# **Източници**

**[1]** threejs

https://github.com/mrdoob/three.js/

**[2]** physijs

https://github.com/chandlerprall/Physijs

**[3]** Mozilla Web Events

<https://developer.mozilla.org/en-US/docs/Web/Events>

**[4]** Netlify

<https://www.netlify.com/>

**[5]** Chrome DevTools

https://developer.chrome.com/docs/devtools