Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Современные платформы программирования (СПП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

«Игровое веб-приложение Гонки»

БГУИР КП 1-40 01 01 114 ПЗ

Студент: гр. 151001 Матюшенко В. А.

Руководитель: асс. Швед Е. И.

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5

1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ 6

2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 8

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 9

4 КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 12

5 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 16

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПРОГРАММЫ 18

ЗАКЛЮЧЕНИЕ 21

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 22

ПРИЛОЖЕНИЕ А 23

Обобщённый алгоритм отрисовки кадра гонки 23

ПРИЛОЖЕНИЕ Б 24

Исходный код модуля game.js 24

# ВВЕДЕНИЕ

На сегодняшний день одним из наиболее востребованных направлением в программировании является разработка компьютерных игр. В частности, браузерные игры на основе технологий, таких как Node.js, становятся все более популярными благодаря своей доступности и возможности многопользовательского взаимодействия в реальном времени. Для получения опыта в этой области в качестве темы моего курсового проекта я выбрал разработку браузерной гоночной игры, так как именно этот жанр до сих пор является одним из самых часто играемых. Моя цель заключается в разработке программного средства, которое будет визуализировать весь процесс игры в гонки. В процессе разработки будет рассмотрен ряд ключевых аспектов, включая выбор архитектуры приложения, реализацию серверной логики с использованием Node.js, а также создание клиентской части с применением современных веб-технологий.

В настоящей пояснительной записке отражены следующие этапы написания курсового проекта:

1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству.

2 Анализ требований к программному средству и разработка функциональных требований.

3 Проектирование программного средства.

4 Конструирование программного средства.

5 Тестирование, проверка работоспособности и анализ полученных результатов.

6 Руководство пользователя программы.

# 1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

Одним из прототипов программного средства является компьютерная игра Need For Speed Most Wanted. Данная игра позволяет игроку создавать автомобили для участия в гонках. Целью гонок явялется придти первым или, другими словами, иметь наименьшее время прохождения какого-то маршрута. Элементы игрового мира обычно представляют собой сам автомобиль, трассу, спидометр, карту и процент прохождения гонки. Особенностями игры являются реалистичная механика и графика.



Рисунок 1.1 – Игровой мир Need For Speed

В качестве другого прототипа была рассмотрена более старая игра Pole Position Namco/Atari. Игру часто считают одной из самых важных в истории гоночных игр, поскольку она была одной из первых, принявших вид автомобиля сзади, устанавливая стандарт, которому будут следовать многие более поздние игры этого жанра. Такая перспектива обеспечивала более захватывающее ощущение пребывания за рулем. В сравнении с первым прототипом в данной игре отсутствует детализация текстур и реалистичная механика, но цель игры остается той же. В то же время игра требует меньше ресурсов для запуска.



Рисунок 1.2 – Игровой мир Pole Position

Для получения необходимой для работы информации я использовал следующие литературные источники:

1 Документация по HTML.

2 Документация по JavaScript.

3 Документация по Node.js.

Было решено, что программное средство будет включать в себя главное меню, игровой мир и таблицу рекордов. Главное меню будет состоять из двух пунктов: начать гонку и посмотреть таблицу рекордов. При выборе пункта «начать гонку» отрисуется игровой мир, состоящий из автомобиля, дороги, препятствий, секундомера, спидометра и процента прохождения гонки. Цель игрока – проехать гонку за наименьшее время, входя в повороты и минуя препятствия, которые замедляют скорость. Для управления игроку доступны клавиши стрелки, соответствующие направлению движения автомобиля. После прохождения гонки игроку показывается его время. Лучшее время заносится в таблицу рекордов. Таблицу рекордов можно просмотреть через главное меню. Чтобы упростить процесс, отрисовка будет происходить с использованием спрайтов.

# 2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

После анализа прототипа программного средства были сформированы вышеупомянутые требования. Исходя из поставленной задачи и сформулированных требований, необходимо реализовать следующие функции:

1 Понятная визуализация главного меню и игрового мира.

2 Отрисовка игрового мира и его элементы (автомобиль, дорога, препятствия, секундомер, спидометр и процент прохождения гонки) при выборе пункта меню начать гонку.

3 Генерация трассы определенной длины с поворотами, возвышенностями и препятствиями, размещенными случайным образом.

4 Отрисовка таблицы рекордов при выборе пункта меню посмореть таблицу рекордов.

5 Визуализация движения автомобиля при нажатии соответствующих клавиш управления.

6 Обновление значений секундомера, спидометра и процента прохождения гонки.

7 Показ времени прохождения гонки после ее прохождения.

8 Сохранение лучших десяти значений времени прохождения гонок в таблице рекордов.

9 Запуск приложения через браузеры Chrome, Яндекс, Opera, Edge, Firefox и Safari.

Программное средство необходимо спроектировать таким образом, чтобы оно удовлетворяло всем полученным функциональным требованиям. Для этого программное средство должно быть браузерным приложением, которое может быстро обрабатывать действия пользователя и мгновенно визуализировать полученный результат.

# 3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Целью проектирования является создания браузерного приложения для гоночной игры. Приложение организовано с использованием разделения логики на серверную и клиентскую части. Серверная часть обслуживает запросы клиента и статические файлы. Также ее ролью являются обеспечение запуска приложения и хранения таблицы рекордов. В свою очередь клиентская часть отображает интерфейс пользователя и анимации, обеспечивает управление игрой и реализует основную механику игры.

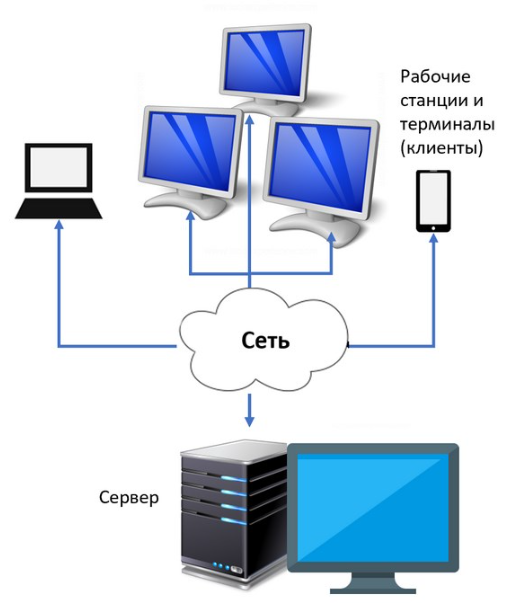


Рисунок 3.1 – [Архитектура «Клиент-Сервер»](https://itelon.ru/blog/arkhitektura-klient-server/" \t "https://www.google.com/_blank)

После запуска приложения пользователю будет показано главное меню, из которого он может начать гонку или перейти в меню с таблицей рекордов. При переходе в меню с таблицей рекордов будет происходить загрузка данных о рекордах с сервера. Перед началом гонки переинициализируются необходимые переменные и сгенерируется новая трасса. В интерфейсе пользователя во время гонки будут обновляться значения секундомера, спидометра и процента прохождения гонки. Пользователь может в любой момент вернуться в меню или завершить гонку. При завершении гонки время гонки отправляется на сервер и записывается в список рекордов.

Главными алгоритмами программного средства являются алгоритм добавления времени в список рекордов, алгоритм генерации трассы и алгоритм отрисовки кадра гонки.

Основной идеей алгоритма добавления времени в список рекордов является сортировка массива с добавленным временем по возрастанию с последующим ограничением записей в массиве до 10. Тем самым сохраняются только 10 наименьших значений времени. Также значение из тела запроса валидируется для корректной работы приложения. В конце алгоритма клиенту отправляется соответствующий ответ.

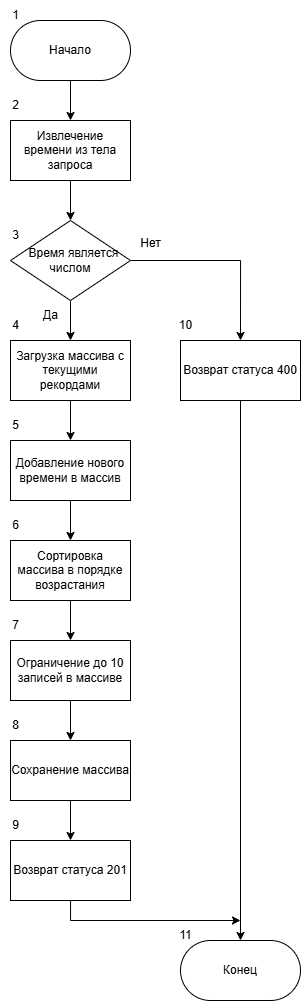


Рисунок 3.2 – Обобщенный алгоритм добавления времени

Целью алгоритма генерации трассы является создание псевдослучайной трассы. Трасса состоит из последовательных зон с различной высотой, кривизной и элементами окружения. Зоны для простоты делятся на элементарные сегменты. Высота и кривизна трассы изменяются на основе текущего состояния. Для этого используется матрицы переходов, чтобы определять вероятности перехода из одного состояния в другое. Для достижения сглаженного перехода между зонами применяется синусоидальная функция. Элементы окружения генерируются с заданной вероятностью.



Рисунок 3.3 – Обобщенный алгоритм генерации трассы

В алгоритме отрисовки кадра гонки происходит обработка клавиш ввода, обновление состояния игрока, проверка условий завершения игры, а также отрисовка фона, трассы, автомобиля и интерфейса. Обобщенный алгоритм отрисовки кадра гонки находится в приложении А.

4 КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Языком программирования для написания программного средства был выбран язык JavaScript на движке Node.js. Node.js основан на движке V8 от Google Chrome, который обеспечивает высокую скорость выполнения кода на JavaScript. Также Node.js позволяет создавать серверные приложения, используя тот же язык, что и для клиентской части, что делает его особенно популярным для веб-разработки. Дополнительно Node.js имеет богатую экосистему библиотек и инструментов, доступных через npm (Node Package Manager), благодаря чему разработка значительно ускоряется.  
 В добавок к этому, для создания структуры веб-страницы использовался стандартный язык разметки HTML, а для оформления применялся язык стилей CSS. Для упрощенной работы с JavaScript используется библиотека jQuery.

В качестве среды разработки была выбрана среда PhpStorm. Несмотря на то, что среда PhpStorm создана специально для работы с PHP, она также поддерживает множество других технологий, включая HTML, CSS, JavaScript и популярные фреймворки. Также в PhpStorm присутствуют инструменты для работы с системой контроля версий Git, которая также использовалась при написании приложения.

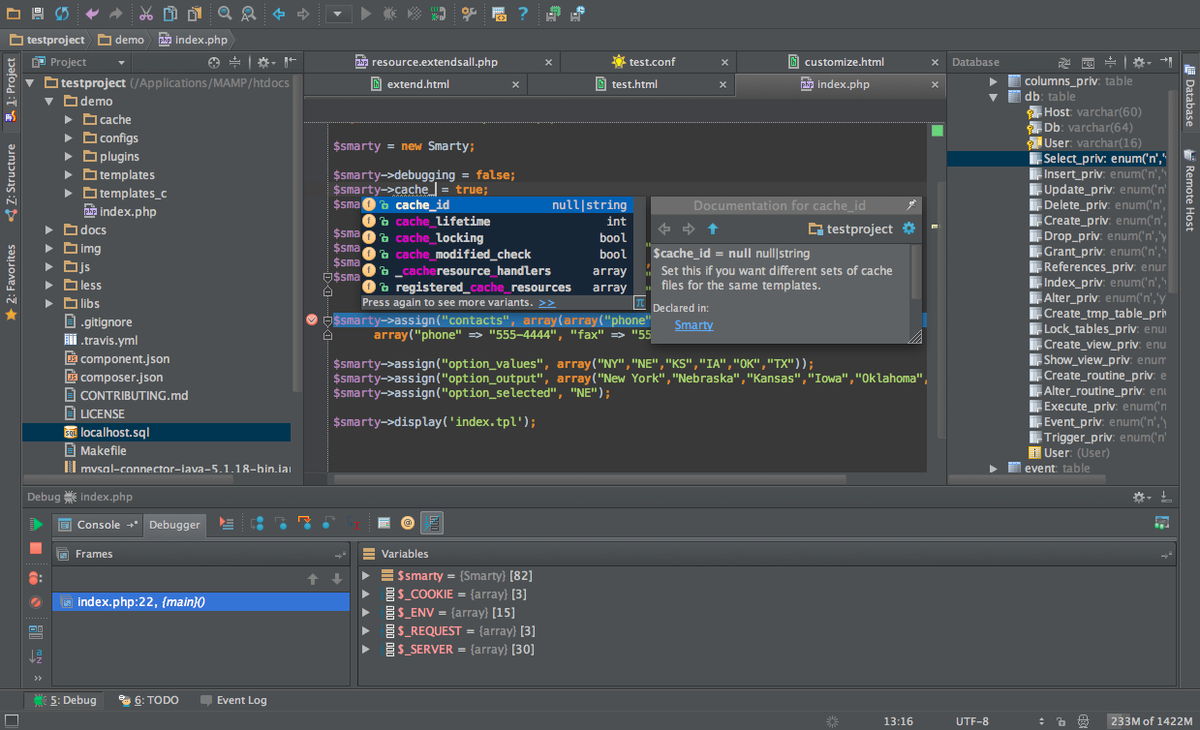


Рисунок 4.1 – Интегрированная среда разработки PhpStorm

Сам проект состоит из следующих конмпонентов:

1 Директория node\_modules. Это стандартная директория для зависимостей, установленных через npm.

2 Директория public. Это директория для статических файлов, таких как: icon.ico (иконка для браузера), index.html, main.js, spritesheet.png (спрайтовая картинка для графики игры), style.css.

3 Директория src. Это директория с исходным кодом, содержит модули: api.js, game.js, render.js, road.js, utils.js.

4 Корневые файлы. Файлы package.json и package-lock.json являются конфигурационными файлами npm, описывают зависимости и скрипты. В файле records.json хранятся данные о рекордах. Файл server.js необходим для запуска приложения.

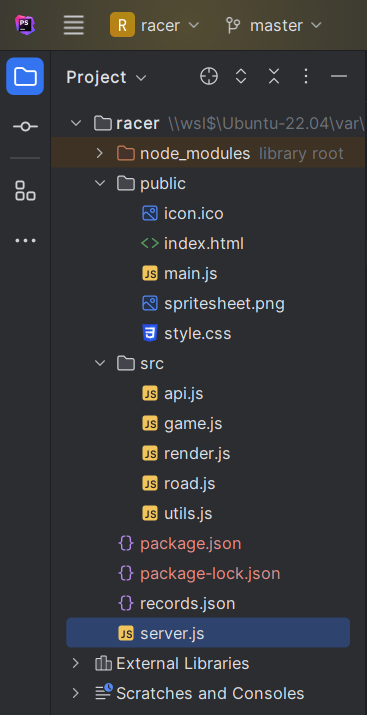


Рисунок 4.2 – Структура проекта

Файл index.html добавляет иконку для вкладки браузера из файла icon.ico, подключает файл style.css для оформления элементов страницы, подключает библиотеку jQuery из внешнего CDN и подключает основной JavaScript-файл main.js. В нем также объявлен HTML-элемент canvas для отрисовки графики игры.

В файле main.js импортируется класс Game из файла game.js. Когда DOM полностью загружен, создаётся новый объект игры на основе импортированного класса Game и через метод start() игра запускается.

Чтобы структурировать код и удобно отделить логику работы, программа была разделена на следующие модули:

1 api.js. Здесь объявлены функции для получения данных из файла records.json для таблицы рекордов и добавления времени в файл records.json через API запросы. Для передачи данных используется формат JSON.

2 utils.js. Здесь объявлены функции для инициализации игрового окна, изменения его размеров, отслеживания нажатых клавиш и конвертации времени в строку формата минуты:секунды:миллисекунды.

3 road.js. Здесь объявлена функция для генерации трассы по заданным параметрам. Сама трасса имеет повороты и холмы. Также случайным образом генерируются игровые объекты вне трассы для создания более реалистичной обстановки.

4 render.js. Здесь объявлены функции для отображения строк, спрайтов, а также для отрисовки сегментов трассы.

5 game.js. Здесь объявлен класс, который, используя вышеописанные модули, инициализирует игру, управляет процессом игры, занимается отрисовкой и обрабатывает нажатые пользователем клавиши. В нем объявлены необходимые константы и функции для отрисовки главного меню, таблицы рекордов и кадра гонки.

6 server.js. В этом файле находится серверное приложение на Node.js с использованием фреймворка Express. В нем инициализируется сервер Express, устанавливает порт, на котором будет работать сервер. Функции Express позволяют серверу обрабатывать данные в формате JSON и делают директории public и src доступными для статических файлов. Здесь также объявлены функции обработки обычных и API запросов и функция для запуска сервера.

Дополнительно были скачаны зависимости из файла package.json. Фреймворк Express нужен для быстрого и удобного создания веб-сервера. В качестве модуля для работы с путями файлов и директорий был выбрал модель Path. Также необходим модуль для работы с файловой системой, а именно для чтения и записи файлов. Дополнительно в файле указан скрипт для запуска сервера с помощью команды node server.js.

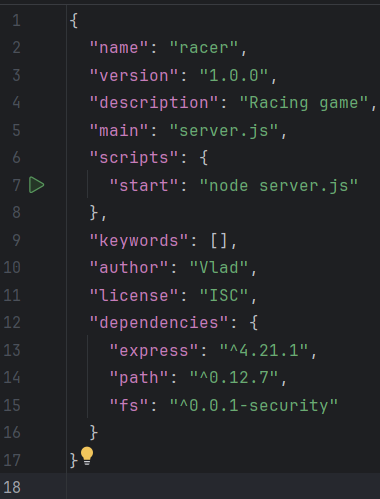


Рисунок 4.3 – Содержимое файла package.json

Таким образом после запуска сервера пользователю отображается файл index.html, который в свою очередь подгружает необходимые стили и скрипты. В файле main.js создается новый объект игры класса из модуля game.js, который инициализирует нужные переменные и игровое окно с помощью модуля utils.js и отрисовывает пользователю главное меню, используя модуль render.js. Спрайты для всех элементов игрового мира и интерфейса берутся из файла spritesheet.png. Для загрузки таблицы рекордов, а также для добавления нового времени в список рекордов, используется модуль api.js. Перед началом отрисовки кадров гонки генерируется новая трасса через модуль road.js. В интерфейсе пользователя присутствуют подсказки. Трасса отрисовывается с учетом перспективы, что делает игру более приближенной к реальной гонке.

# 5 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Для выявления ошибок на этапе конструирования программного средства было проведено полное тестирование. Специфики тестирования и результаты отражены в таблице 5.1.

Таблица 5.1 – Тестирование программного средства

|  |  |  |
| --- | --- | --- |
| Специфика тестирования | Ожидаемый результат | Полученный результат |
| Запуск приложения и перезагрузка страницы | Открытие главного меню. Значения в таблице рекордов сохраняются. | Тест пройден |
| Нажатие клавиши F11 в любой момент | Переход в полноэкранный режим или обратно. | Тест пройден |
| Нажатие клавиши Tab в главном меню | Отображение 10 записей с временем из таблицы рекордов в формате минуты:секунды:  миллисекунды отсортированными по возрастанию. | Тест пройден |
| Нажатие клавиши Escape при показе таблицы рекордов | Открытие главного меню. | Тест пройден |
| Нажатие клавиши Space в главном меню | Инициализация переменных игрока, генерация трассы и ее объектов, отображение игрового мира. | Тест пройден |
| Нажатие клавиш поворота автомобиля | Смещение автомобиля в соответствующую сторону, отрисовка соответствующего спрайта автомобиля. | Тест пройден |
| Нажатие клавиши ускорения автомобиля на трассе | Увеличение скорости автомобиля до максимальной скорости. | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |
| --- | --- | --- |
| Нажатие клавиши ускорения автомобиля вне трасса | Медленное увеличение скорости автомобиля до максимальной скорости вне трассы. | Тест пройден |
| Управление автомобилем | Соответствующие изменения показателей секундомера, спидометра и процента прохождения гонки. | Тест пройден |
| Завершение гонки | Отображение значения 100 в проценте прохождения гонки. Прекращение изменения показателей секундомера и спидометра. Остановка автомобиля после белой полосы. Запись значения секундомера в список рекордов. | Тест пройден |
| Не завершать гонку в течение 10 минут | Прекращение изменения показателей секундомера, спидометра и процента прохождения гонки. Остановка автомобиля. | Тест пройден |
| Нажатие клавиши Escape во время гонки | Открытие главного меню. | Тест пройден |

Проведённые тесты показали корректное выполнение всех требуемых действий. В ходе итогового тестирования не было выявлено каких-либо ошибок или некорректной работы программного средства.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПРОГРАММЫ

Для запуска приложения необходимо в корневой директории проекта выполнить команду node server.js. По умолчанию приложение будет доступно по ссылке <http://localhost:3000/>. При открытии приложения пользователю доступно главное меню. Из него он может начать гонку при нажатии клавиши Space, просмотреть таблицу рекордов при нажатии Tab и перейти в полноэкранный режим при нажатии F11 (стандартная функция браузера).

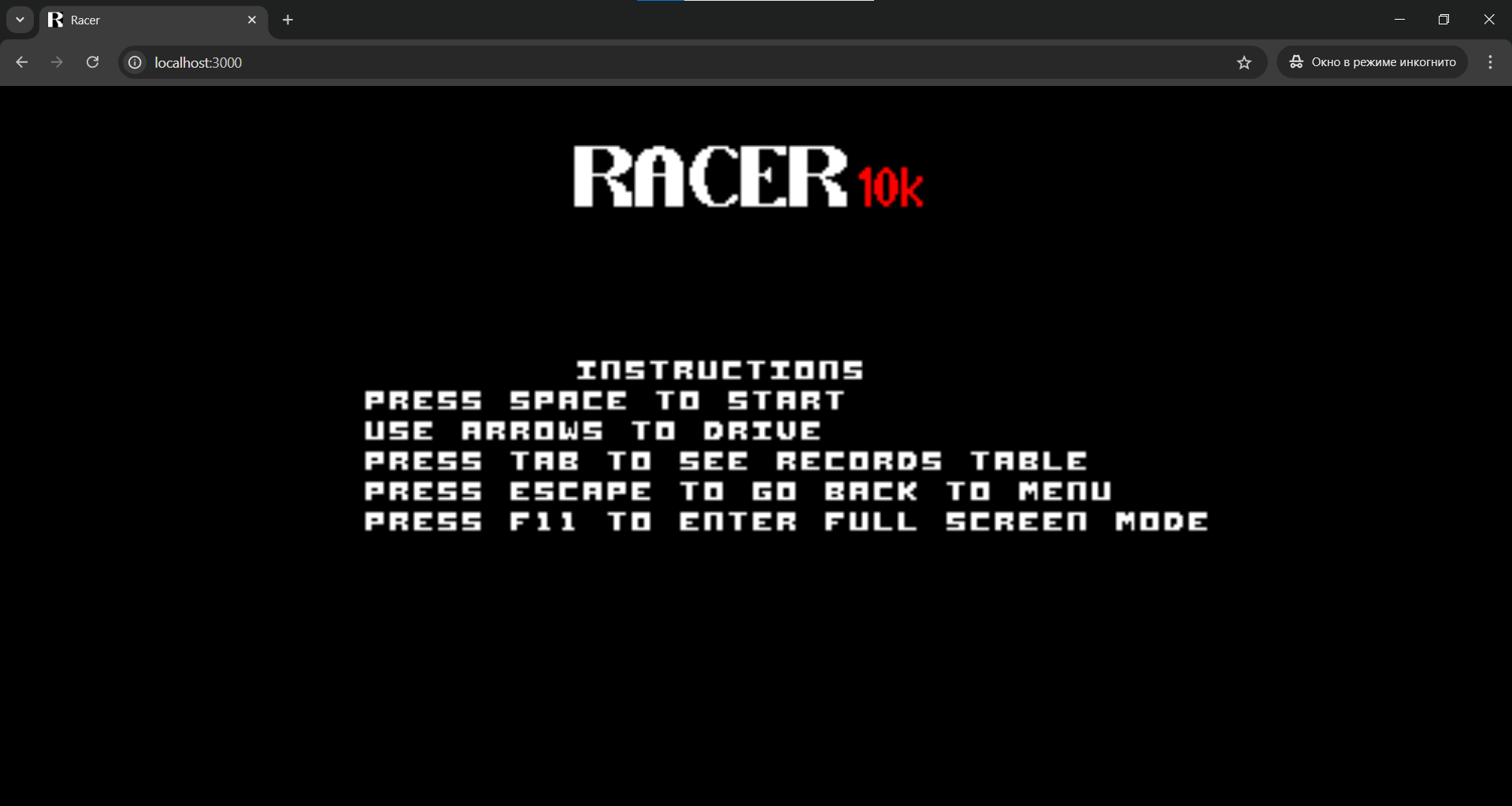


Рисунок 6.1 – Главное меню

В таблице рекордов показаны 10 строк с наименьшим временем заезда. Когда гонка завершается, время добавляется в существующий список, список сортируется и 10 первых записей сохраняются. Если в списке меньше 10 записей, то оставшиеся строки таблицы заполняются текстом «NO TIME». Время хранится в формате минуты:секунды:миллисекунды. Вернутся в главное меню можно при нажатии клавиши Escape.



Рисунок 6.2 – Таблица рекордов в полноэкранном режиме

Когда начинается гонка, на экране помимо трассы с препятствиями и автомобилем появляются секундомер, спидометр и процент прохождения гонки. Время в секундомере находится в левом верхнем углу и отображает время в формате минуты:секунды:миллисекунды. Под секундомером находится спидометр, который показывает скорость в километрах в час. В правом верхнем углу находится процент прохождения гонки.

Ускорение автомобиля происходит при нажатии клавиши Up Arrow, поворот влево и вправо при нажатии Left Arrow и Right Arrow соответственно. Если автомобиль съезжает с трассы на песок, его ускорение, как и максимальная скорость, значительно уменьшается. В любой момент гонки при нажатии клавиши Escape можно вернутся в гланое меню. В таком случае время заезда не сохранится, но при начале новой гонки все необходимые переменные переинициализируются и сгенерируется новая трасса.



Рисунок 6.3 – Кадр гонки в полноэкранном режиме

Гонка завершается, когда процент прохождения гонки достигает значения 100. На трассе это отображается белой полосой. После этого время добавляется в список таблицы рекордов. Чтобы вернутся в главное меню, нужно нажать клавишу Escape.



Рисунок 6.4 – Завершенная гонка в полноэкранном режиме

Также гонка считается завершенной, если время заезда составляет более 10 минут.

# ЗАКЛЮЧЕНИЕ

В пояснительно записке я отразил все этапы разработки моего программного средства. На каждом из этих этапов я постарался показать ход моих мыслей и объяснить причины принятых решений. В результате работы над курсовым проектом было сконструировано и разработано программное средство для игры в гонки. Разработанное программное средство в полной мере выполняет все операции, заданные в условии курсового проекта, работает корректно и без сбоев.

После анализа приведённого аналога и выявления необходимых требований, было спроектировано рассмотренное программное средство. Для его создания было проведено углублённое ознакомление с предметной областью и были выбраны наиболее оптимальные решения. В ходе тестирования было показано, что программное средство не подведёт пользователя ни при каких обстоятельствах.

К сожалению, из-за кратких сроков пришлось отложить реализацию разных режимов гонок и мультиплеерной игры. Но несмотря на это, проделанная работа принесла мне много опыта, необходимого в моей будущей профессии.

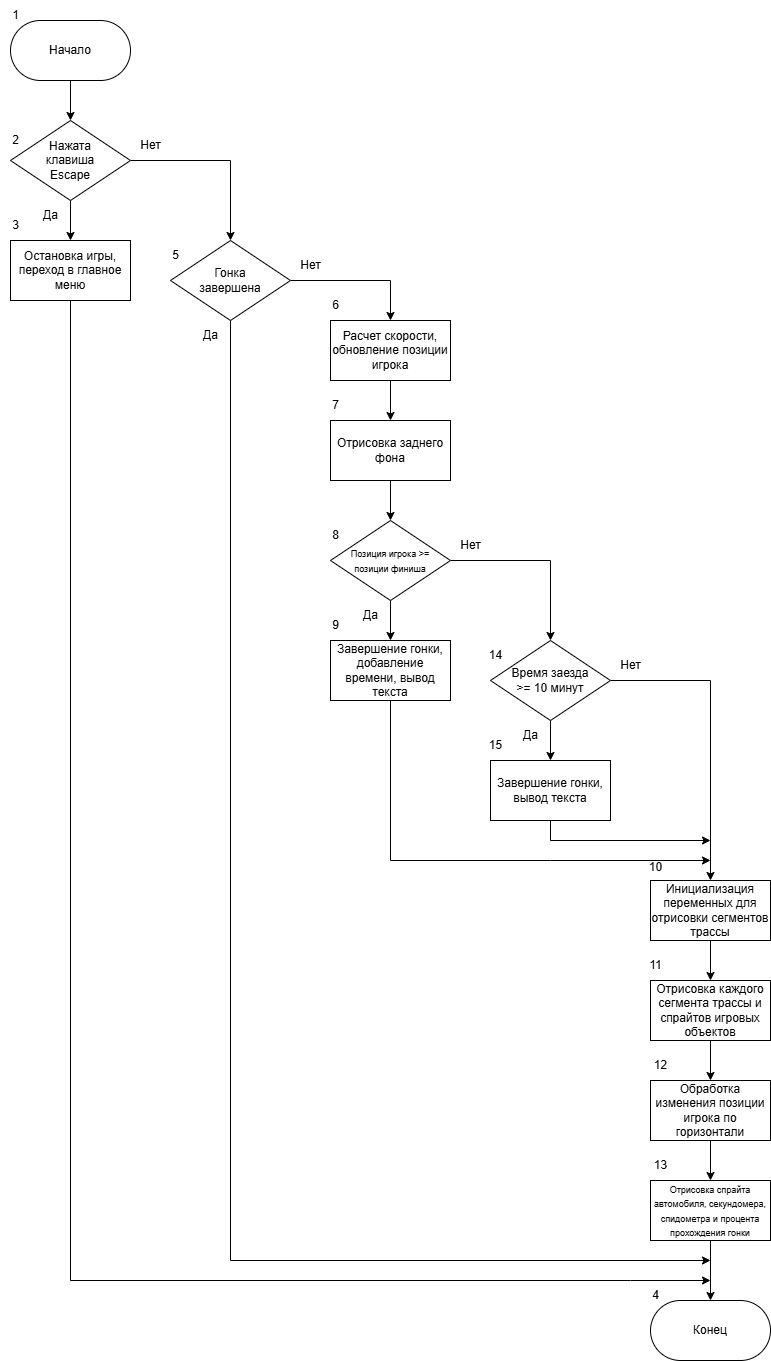
Работа была разделена на этапы, такие как анализ существующих решений, постановка требований к проектируемому программному средству, разработка алгоритма и его схемы, конструирование программного средства, отладка и тестирование. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее программное средство.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://developer.mozilla.org/en-US/docs/Web/HTML [Электронный ресурс] – Документация по HTML.
2. https://developer.mozilla.org/en-US/docs/Web/JavaScript [Электронный ресурс] – Документация по JavaScript.
3. https://nodejs.org/docs/latest/api/ [Электронный ресурс] – Документация по Node.js.

# ПРИЛОЖЕНИЕ А

## Обобщённый алгоритм отрисовки кадра гонки



# ПРИЛОЖЕНИЕ Б

## Исходный код модуля game.js

import { init as initUtils, timeToString } from './utils.js';  
import { getRecords, addRecord } from './api.js';  
import { drawString, drawImage, drawSprite, drawSegment } from './render.js';  
import { generateRoad } from './road.js';  
  
export class Game {  
 constructor() {  
 this.CANVAS\_SELECTOR = '#gameCanvas';  
 this.SPRITESHEET\_NAME = 'spritesheet.png';  
 this.TIMEOUT = 30;  
  
 this.KEY = {  
 SPACE: 32,  
 ESCAPE: 27,  
 TAB: 9,  
 ARROW\_UP: 38,  
 ARROW\_DOWN: 40,  
 ARROW\_LEFT: 37,  
 ARROW\_RIGHT: 39,  
 };  
   
 this.ROAD = {  
 LENGTH: 2500,  
 ZONE\_SIZE: 250,  
 MAX\_HEIGHT: 900,  
 MAX\_CURVE: 400,  
 CURVE\_PROBABILITY: 0.8,  
 MOUNTAIN\_PROBABILITY: 0.8,  
 SEGMENT\_SIZE: 5,  
 SEGMENT\_PER\_COLOR: 4,  
 };  
  
 this.RENDER = {  
 WIDTH: 320,  
 HEIGHT: 240,  
 DEPTH: 150,  
 CAMERA\_DISTANCE: 30,  
 CAMERA\_HEIGHT: 100,  
 };  
  
 this.PLAYER = {  
 START\_POSITION: 10,  
 POSITION\_X\_OFFSET\_OFF\_ROAD: 130,  
 MAX\_SPEED: 15,  
 MAX\_SPEED\_OFF\_ROAD: 3,  
 MAX\_SPEED\_KILOMETERS: 200,  
 MAX\_TIME\_MINUTES: 10,  
 ACCELERATION: 0.05,  
 DECELERATION: 0.1,  
 DECELERATION\_OFF\_ROAD: 0.2,  
 BREAKING: 0.5,  
 TURNING: 5.0,  
 };  
  
 this.COLOR = {  
 WHITE: "#fff",  
 GREY: "#777",  
 DARK\_GREY: "#999",  
 RED: "#e00",  
 SANDY: "#eda",  
 DARK\_SANDY: "#dc9",  
 }  
  
 this.LOGO = {x: 357, y: 9, w: 115, h: 20};  
 this.BACKGROUND = {x: 0, y: 9, w: 320, h: 120};  
 this.CACTUS = {x: 321, y: 9, w: 23, h: 50};  
 this.POLE = {x: 345, y: 9, w: 11, h: 14};  
 this.CAR = {  
 STRAIGHT: {x: 0, y: 130, w: 69, h: 38},  
 LEFT: {x: 70, y: 130, w: 77, h: 38},  
 RIGHT: {x: 148, y: 130, w: 77, h: 38}  
 };  
  
 this.menuInterval = null;  
 this.gameInterval = null;  
 this.tableInterval = null;  
  
 this.spritesheet = null;  
 this.context = null;  
 this.keys = [];  
 this.records = [];  
  
 this.road = [];  
 this.player = {  
 position: 0,  
 positionX: 0,  
 lastPositionXOffset: 0,  
 speed: 0,  
 startTime: null,  
 finished: false,  
 };  
 }  
  
 start() {  
 const self = this;  
  
 initUtils(self);  
  
 self.spritesheet = new Image();  
 self.spritesheet.onload = function () {  
 self.menuInterval = setInterval(() => self.renderMenuFrame(), self.TIMEOUT);  
 };  
 self.spritesheet.src = this.SPRITESHEET\_NAME;  
 }  
  
 renderMenuFrame() {  
 const self = this;  
  
 self.context.fillStyle = "rgb(0,0,0)";  
 self.context.fillRect(0, 0, self.RENDER.WIDTH, self.RENDER.HEIGHT);  
 drawImage(self, self.LOGO, 100, 20);  
  
 drawString(self, "Instructions", {x: 100, y: 90});  
 drawString(self, "Press Space to start", {x: 30, y: 100});  
 drawString(self, "Use arrows to drive", {x: 30, y: 110});  
 drawString(self, "Press Tab to see records table", {x: 30, y: 120});  
 drawString(self, "Press Escape to go back to menu", {x: 30, y: 130});  
 drawString(self, "Press F11 to enter full screen mode", {x: 30, y: 140});  
  
 if (self.keys[self.KEY.SPACE]) {  
 self.keys[self.KEY.SPACE] = false;  
 clearInterval(self.menuInterval);  
  
 self.road = [];  
  
 self.player.position = self.PLAYER.START\_POSITION;  
 self.player.positionX = 0;  
 self.player.speed = 0;  
 self.player.lastPositionXOffset = 0;  
 self.player.finished = false;  
  
 generateRoad(self);  
  
 self.gameInterval = setInterval(() => self.renderGameFrame(), self.TIMEOUT);  
 self.player.startTime = new Date();  
 }  
  
 if (self.keys[self.KEY.TAB]) {  
 self.keys[self.KEY.TAB] = false;  
 clearInterval(self.menuInterval);  
  
 getRecords().then(records => {  
 self.records = records;  
 });  
  
 self.tableInterval = setInterval(() => self.renderTableFrame(), self.TIMEOUT);  
 }  
 }  
  
 renderTableFrame() {  
 const self = this;  
  
 self.context.fillStyle = "rgb(0,0,0)";  
 self.context.fillRect(0, 0, self.RENDER.WIDTH, self.RENDER.HEIGHT);  
 drawImage(self, self.LOGO, 100, 20);  
  
 drawString(self, "Records", {x: 100, y: 90});  
 for (let i = 0; i < 10; i++) {  
 drawString(self,  
 (i + 1) + " - " + (self.records[i] ? timeToString(self.records[i]) : "no time"),  
 {x: 100 - (i === 9 ? 8 : 0), y: 100 + i \* 10}  
 );  
 }  
 drawString(self, "Press Escape to go back to menu", {x: 30, y: 210});  
  
 if (self.keys[self.KEY.ESCAPE]) {  
 self.keys[self.KEY.ESCAPE] = false;  
 clearInterval(self.tableInterval);  
  
 self.menuInterval = setInterval(() => self.renderMenuFrame(), self.TIMEOUT);  
 }  
 }  
  
 renderGameFrame() {  
 const self = this;  
  
 if (self.keys[self.KEY.ESCAPE]) {  
 self.keys[self.KEY.ESCAPE] = false;  
 clearInterval(self.gameInterval);  
  
 self.menuInterval = setInterval(() => self.renderMenuFrame(), self.TIMEOUT);  
  
 return;  
 }  
  
 if (self.player.finished) {  
 return;  
 }  
  
 if (Math.abs(self.player.lastPositionXOffset) > self.PLAYER.POSITION\_X\_OFFSET\_OFF\_ROAD  
 && self.player.speed > self.PLAYER.MAX\_SPEED\_OFF\_ROAD  
 ) {  
 self.player.speed -= self.PLAYER.DECELERATION\_OFF\_ROAD;  
 }  
  
 if (self.keys[self.KEY.ARROW\_UP]) {  
 self.player.speed += self.PLAYER.ACCELERATION;  
 } else if (self.keys[self.KEY.ARROW\_DOWN]) {  
 self.player.speed -= self.PLAYER.BREAKING;  
 } else {  
 self.player.speed -= self.PLAYER.DECELERATION;  
 }  
  
 self.player.speed = Math.max(self.player.speed, 0);  
 self.player.speed = Math.min(self.player.speed, self.PLAYER.MAX\_SPEED);  
 self.player.position += self.player.speed;  
  
  
 self.context.fillStyle = self.COLOR.DARK\_SANDY;  
 self.context.fillRect(0, 0, self.RENDER.WIDTH, self.RENDER.HEIGHT);  
  
 let backgroundX = -self.player.positionX / 2 % (self.BACKGROUND.w);  
 drawImage(self, self.BACKGROUND, backgroundX - self.BACKGROUND.w + 1, 0);  
 drawImage(self, self.BACKGROUND, backgroundX + self.BACKGROUND.w - 1, 0);  
 drawImage(self, self.BACKGROUND, backgroundX, 0);  
  
  
 let absoluteIndex = Math.floor(self.player.position / self.ROAD.SEGMENT\_SIZE);  
 let time = (new Date()).getTime() - self.player.startTime.getTime();  
  
 if (absoluteIndex >= self.ROAD.LENGTH - self.RENDER.DEPTH - 1) {  
 self.player.finished = true;  
  
 addRecord(time);  
  
 drawString(self, "You did it!!!", {x: 100, y: 30});  
 drawString(self, "Press Escape to go back to menu", {x: 30, y: 40});  
 } else if (Math.floor(time / 60000) >= self.PLAYER.MAX\_TIME\_MINUTES) {  
 self.player.finished = true;  
  
 drawString(self, "Too slow, try again", {x: 100, y: 30});  
 drawString(self, "Press Escape to go back to menu", {x: 30, y: 40});  
 }  
  
 let spriteBuffer = [];  
 let currentSegmentIndex = (absoluteIndex - 2) % self.ROAD.LENGTH;  
 let currentSegmentPosition = (absoluteIndex - 2) \* self.ROAD.SEGMENT\_SIZE - self.player.position;  
 let currentSegment = self.road[currentSegmentIndex];  
 let lastProjectedHeight = Number.POSITIVE\_INFINITY;  
 let probedDepth = 0;  
 let counter = absoluteIndex % (2 \* self.ROAD.SEGMENT\_PER\_COLOR);  
 let playerPosSegmentHeight = self.road[absoluteIndex % self.ROAD.LENGTH].height;  
 let playerPosNextSegmentHeight = self.road[(absoluteIndex + 1) % self.ROAD.LENGTH].height;  
 let playerPosRelative = (self.player.position % self.ROAD.SEGMENT\_SIZE) / self.ROAD.SEGMENT\_SIZE;  
 let playerHeight = self.RENDER.CAMERA\_HEIGHT + playerPosSegmentHeight + (playerPosNextSegmentHeight - playerPosSegmentHeight) \* playerPosRelative;  
  
 let baseOffset = currentSegment.curve + (self.road[(currentSegmentIndex + 1) % self.ROAD.LENGTH].curve - currentSegment.curve) \* playerPosRelative;  
  
 self.player.lastPositionXOffset = self.player.positionX - baseOffset \* 2;  
  
 let iter = self.RENDER.DEPTH;  
 while (iter--) {  
 let nextSegmentIndex = (currentSegmentIndex + 1) % self.ROAD.LENGTH;  
 let nextSegment = self.road[nextSegmentIndex];  
  
 let startProjectedHeight = Math.floor((playerHeight - currentSegment.height) \* self.RENDER.CAMERA\_DISTANCE / (self.RENDER.CAMERA\_DISTANCE + currentSegmentPosition));  
 let startScaling = 30 / (self.RENDER.CAMERA\_DISTANCE + currentSegmentPosition);  
  
 let endProjectedHeight = Math.floor((playerHeight - nextSegment.height) \* self.RENDER.CAMERA\_DISTANCE / (self.RENDER.CAMERA\_DISTANCE + currentSegmentPosition + self.ROAD.SEGMENT\_SIZE));  
 let endScaling = 30 / (self.RENDER.CAMERA\_DISTANCE + currentSegmentPosition + self.ROAD.SEGMENT\_SIZE);  
  
 let currentHeight = Math.min(lastProjectedHeight, startProjectedHeight);  
 let currentScaling = startScaling;  
  
 if (currentHeight > endProjectedHeight) {  
 drawSegment(  
 self,  
 self.RENDER.HEIGHT / 2 + currentHeight,  
 currentScaling, currentSegment.curve - baseOffset - (self.player.positionX - baseOffset \* 2) \* currentScaling,  
 self.RENDER.HEIGHT / 2 + endProjectedHeight,  
 endScaling,  
 nextSegment.curve - baseOffset - (self.player.positionX - baseOffset \* 2) \* endScaling,  
 counter < self.ROAD.SEGMENT\_PER\_COLOR, currentSegmentIndex === 2 || currentSegmentIndex === (self.ROAD.LENGTH - self.RENDER.DEPTH));  
 }  
 if (currentSegment.sprite) {  
 spriteBuffer.push({  
 y: self.RENDER.HEIGHT / 2 + startProjectedHeight,  
 x: self.RENDER.WIDTH / 2 - currentSegment.sprite.pos \* self.RENDER.WIDTH \* currentScaling + currentSegment.curve - baseOffset - (self.player.positionX - baseOffset \* 2) \* currentScaling,  
 ymax: self.RENDER.HEIGHT / 2 + lastProjectedHeight,  
 s: 2.5 \* currentScaling,  
 i: currentSegment.sprite.type  
 });  
 }  
  
 lastProjectedHeight = currentHeight;  
  
 probedDepth = currentSegmentPosition;  
  
 currentSegmentIndex = nextSegmentIndex;  
 currentSegment = nextSegment;  
 currentSegmentPosition += self.ROAD.SEGMENT\_SIZE;  
  
 counter = (counter + 1) % (2 \* self.ROAD.SEGMENT\_PER\_COLOR);  
 }  
  
 let sprite;  
 while (sprite = spriteBuffer.pop()) {  
 drawSprite(self, sprite);  
 }  
  
 let carSprite;  
 if (self.keys[self.KEY.ARROW\_LEFT]) {  
 if (self.player.speed > 0) {  
 self.player.positionX -= self.PLAYER.TURNING;  
 }  
 carSprite = {  
 a: self.CAR.LEFT,  
 x: 117,  
 y: 190  
 };  
 } else if (self.keys[self.KEY.ARROW\_RIGHT]) {  
 if (self.player.speed > 0) {  
 self.player.positionX += self.PLAYER.TURNING;  
 }  
 carSprite = {  
 a: self.CAR.RIGHT,  
 x: 125,  
 y: 190  
 };  
 } else {  
 carSprite = {  
 a: self.CAR.STRAIGHT,  
 x: 125,  
 y: 190  
 };  
 }  
 drawImage(self, carSprite.a, carSprite.x, carSprite.y);  
 drawString(self,  
 timeToString(time),  
 {x: 2, y: 2}  
 );  
 drawString(self,  
 Math.round(self.player.speed / self.PLAYER.MAX\_SPEED \* self.PLAYER.MAX\_SPEED\_KILOMETERS) + "kph",  
 {x: 2, y: 12}  
 );  
 drawString(self,  
 Math.round(absoluteIndex / (self.ROAD.LENGTH - self.RENDER.DEPTH) \* 100) + "%",  
 {x: 286, y: 2}  
 );  
 };  
}