

Документация проекта AR Carpet Game

Обзор проекта

AR Carpet Game - интерактивная игра с машинками на детском ковре с дорогами. Поддерживает три режима: AR (дополненная реальность), TOUCH (сенсорное управление) и GYRO (гироскоп).

Структура проекта

```
ar-carpet-game/
├── index.html      # Главная страница
├── assets/
│   ├── carpet-scan.jpg  # Текстура ковра
│   └── models/        # 3D модели машин
│       ├── Buggy.glb
│       ├── CesiumMilkTruck.glb
│       └── Duck.glb
└── src/
    ├── main.js        # Точка входа
    ├── config.js      # Конфигурация
    ├── ar_webxr.js    # AR режим (WebXR)
    ├── nonAr.js       # TOUCH/GYRO режимы
    ├── cars/
    │   ├── Car.js
    │   └── CarModels.js
    ├── roads/          # Дорожная система
    │   ├── roadNetwork.js
    │   └── road_system.js
    ├── traffic/        # Управление трафиком
    │   └── traffic_manager.js
    └── ui/             # Пользовательский интерфейс
        ├── StartScreen.js
        ├── ModeUI.js
        ├── StatsPanel.js
        └── ControlPanel.js
```

Описание файлов

Корневые файлы

`index.html`

Назначение: Главная HTML страница приложения

Основные элементы:

- Import map для загрузки Three.js и MindAR из CDN
- Стартовый экран с выбором режима (`#start`)
- AR контейнер (`#ar-container`)
- Переключатель режимов (`#mode-ui`)
- Настройки (статистика, управление, инверсия, отладка дорог)

Стили:

- Адаптивный дизайн для мобильных и десктопов
 - Неоновый стиль кнопок (зеленый градиент)
 - Фиксированное позиционирование UI элементов
-

Основные модули

`src/main.js`

Назначение: Точка входа приложения, управление режимами

Функции:

- `run(mode, settings)` - Запуск выбранного режима (AR/TOUCH/GYRO)
- `changeMode(mode)` - Переключение между режимами
- Инициализация стартового экрана и переключателя режимов

Логика:

1. Пользователь выбирает режим на стартовом экране
2. Сохранение режима и настроек в localStorage

- Запуск соответствующего модуля (AR или nonAR)
- При ошибке AR - автоматическое переключение на TOUCH

src/config.js

Назначение: Централизованная конфигурация приложения

Параметры:

- `carScales` - Масштабы 3D моделей машин
 - Buggy: 0.3x
 - Duck: 10x
 - Milk Truck: 8.5x
- `carpet` - Размеры ковра (2.0 x 2.5 м)
- `camera` - Настройки камеры (FOV, позиция)
- `cars` - Количество, скорость, высота машин
- `lighting` - Освещение сцены
- `controls` - Чувствительность управления
- `roads` - Параметры дорог (ширина полосы)

Вспомогательные функции:

- `getCarScale(modelName)` - Получить масштаб модели
 - `updateConfig(path, value)` - Обновить параметр конфигурации
-

Модуль машин

src/cars/Car.js

Назначение: Класс отдельной машины

Свойства:

- `model` - 3D модель (Three.js Group/Mesh)
- `roadNetwork` - Ссылка на дорожную сеть
- `path` - Массив узлов маршрута

- `currentPathIndex` - Текущий сегмент пути
- `progress` - Прогресс на текущем сегменте (0-1)
- `currentSpeed` - Текущая скорость
- `currentLane` - Текущая полоса движения
- `targetRotation` / `currentRotation` - Углы поворота

Методы:

- `spawn(startNode, endNode)` - Размещение машины на дороге
- `update()` - Обновление позиции и поворота (каждый кадр)
- `despawn()` - Удаление машины с дороги
- `setGlobalScale(scale)` - Применение глобального масштаба
- `applyRandomColor()` - Случайный цвет машины
- `smoothstep(t)` - Плавная интерполяция движения

Особенности:

- Адаптивная скорость на поворотах (замедление)
- Плавная интерполяция поворота (без бокового скольжения)
- Движение по правой полосе с учетом смещения

`src/cars/CarModels.js`

Назначение: Загрузчик и менеджер 3D моделей машин

Методы:

- `loadAll()` - Асинхронная загрузка всех GLB моделей
- `getModelByName(name)` - Получить загруженную модель по имени
- `getRandomModel()` - Случайная модель для спавна
- `cloneModel(originalModel)` - Клонирование модели для повторного использования

Используемые модели:

1. `Buggy.glb` - Багги (маленькая гоночная машина)
2. `CesiumMilkTruck.glb` - Молоковоз (грузовик)

3. `Duck.glb` - Утка (игрушечная модель)

Дорожная система

`src/roads/roadNetwork.js`

Назначение: Граф дорожной сети (узлы и соединения)

Структура данных:

- `nodes[]` - Массив узлов (перекрестков) `{x, y, connections[]}`
- `roads[]` - Массив дорог `{start, end, length}`
- `lanes[]` - Массив полос движения `{start, end, direction, offset}`

Методы:

- `addNode(x, y)` - Добавить узел (с проверкой дубликатов)
- `addRoad(startNode, endNode)` - Создать двустороннюю дорогу
 - Автоматически создает 2 полосы (по одной в каждом направлении)
 - Полосы смещены от центра дороги (правостороннее движение)
- `findPath(start, end)` - A* поиск пути между узлами
- `getLane(fromNode, toNode)` - Получить полосу движения
- `getClosestNode(x, y)` - Ближайший узел к координатам
- `getStats()` - Статистика сети

Особенности:

- Правостороннее двустороннее движение
- Смещение полос: `0.02` (1/4 ширины дороги)
- A* алгоритм с эвристикой расстояния

`src/roads/road_system.js`

Назначение: Создание конкретной структуры дорог по ковру

Структура дорог (по carpet-scan.jpg):

1. **3 круговые развязки** (левая сторона ковра):

- Верхняя: (-0.65, 0.85)
- Средняя: (-0.70, 0.0)
- Нижняя: (-0.65, -0.85)
- Радиус каждой: 0.12

2. Основные дороги:

- Верхняя извилистая (S-образная)
- Правая извилистая (волнообразная)
- Нижняя извилистая
- Левая (соединяет развязки)

3. Внутренние соединения:

- От развязок к центру
- Центральные горизонтальные дороги
- Перекрестки

Функция `createRoadNetwork(parent, options)`:

- Создает все узлы и соединения
- Опционально визуализирует дороги (для отладки):
 - Серые дорожные сегменты
 - Белые пунктирные центральные линии
 - Круговые развязки с зеленым центром
 - Желтые круги в центре развязок

Параметры:

- `parent` - Three.js Group для добавления визуализации
 - `options.showRoads` - Показать визуализацию дорог (boolean)
-

🚦 Управление трафиком

`src/traffic/traffic_manager.js`

Назначение: Справн и обновление всех машин на дороге

Свойства:

- `cars[]` - Массив всех машин
- `carModels` - Экземпляр CarModels
- `globalScaleMultiplier` - Общий масштаб всех машин
- `isInitialized` - Флаг инициализации

Методы:

- `init()` - Инициализация (загрузка моделей)
- `spawnCars(count)` - Спавн нескольких машин
 - Распределение: 3 Buggy, 2 Truck, 2 Duck
- `spawnCarWithModel(modelData)` - Спавн конкретной модели
 - Создает новую машину
 - Выбирает случайный маршрут
 - Проверяет успешность спавна
- `update()` - Обновление всех машин (каждый кадр)
 - Проверка коллизий между машинами
 - Обновление позиций
 - Респавн завершивших путь
- `setGlobalScale(scale)` - Изменить масштаб всех машин
- `getStats()` - Статистика (активные машины, в пуле)
- `dispose()` - Очистка (удаление всех машин)

Логика коллизий:

- Проверка расстояния между машинами
 - Временная остановка при сближении
 - Автоматическое возобновление движения
-

🎮 Режимы игры

`src/ar_webxr.js`

Назначение: AR режим с использованием WebXR

Особенности:

- WebXR API для AR (без MindAR)
- Hit-test для определения поверхностей
- Калибровка ковра (масштаб + поворот)
- DOM-overlay для UI поверх AR

Этапы работы:

1. Инициализация:

- Проверка поддержки WebXR
- Создание Three.js сцены с `renderer.xr.enabled = true`
- Подготовка игровой группы (carpetGroup)

2. Размещение ковра:

- Hit-test для поиска горизонтальных поверхностей
- Визуальный маркер (reticle) - зеленое кольцо
- Кнопка "РАЗМЕСТИТЬ КОВЕР"
- Размещение на 1см над поверхностью

3. Калибровка:

- Масштаб: ± 0.05 (диапазон 0.5-2.0x)
- Поворот: $\pm 3^\circ$ ($\text{Math.PI}/60$)
- Кнопка "ГОТОВО" для подтверждения

4. Игра:

- Спавн 7 машин после калибровки
- Обновление машин в render loop
- Hit-test отключается после размещения

UI элементы:

- Debug logger (кнопка 
- Панель калибровки
- Инструкции

- AR кнопка запуска
- StatsPanel и ControlPanel

ИСПРАВЛЕНИЯ V16:

- Hit-test работает только ДО калибровки
- После подтверждения продолжается рендеринг
- Нет зависания после нажатия "ГОТОВО"

`src/nonAr.js`

Назначение: TOUCH и GYRO режимы (без AR)

Режим TOUCH:

- Орбитальная камера вокруг ковра
- Управление мышью:
 - Зажатие ЛКМ + движение = вращение
 - Колесо мыши = зум (1.0 - 6.0)
- Управление тачем:
 - 1 палец = вращение
 - 2 пальца = зум (pinch)

Режим GYRO:

- Управление гироскопом телефона
- Запрос разрешения (iOS 13+)
- Автоматическое вращение камеры по датчикам
- Обработка `(deviceorientation)` событий

Общее:

- Загрузка текстуры ковра (`(carpet-scan.jpg)`)
- Небесный фон (`(0x87CEEB)`)
- Освещение: Ambient + Directional
- Инверсия управления (опция)

Сферическая камера:

```
javascript

radius = 2.5 // Расстояние от центра
theta = 0.5 // Угол азимута
phi = 1.1 // Угол возвышения

position.x = radius * sin(phi) * sin(theta)
position.y = radius * cos(phi)
position.z = radius * sin(phi) * cos(theta)
```

ユーザ интерфейс

src/ui/StartScreen.js

Назначение: Стартовый экран с выбором режима

Функциональность:

- Показ экрана при загрузке
- Кнопки выбора режима (AR/TOUCH/GYRO)
- Чекбоксы настроек:
 - Показать статистику
 - Панель управления машинками
 - Инверсия управления
 - Показать дороги (отладка)
- Сохранение настроек в localStorage
- Скрытие экрана после выбора режима

Метод `initStartScreen(onStart)`:

- `(onStart(mode, settings))` - Callback при выборе режима

src/ui/ModeUI.js

Назначение: Переключатель режимов во время игры

Функциональность:

- Кнопки AR/TOUCH/GYRO в верхней части экрана
- Скрыт по умолчанию
- Показывается после запуска игры
- Перезагрузка страницы при смене режима

Метод `initModeUI(onModeChange)`:

- `onModeChange(mode)` - Callback при смене режима

`src/ui/StatsPanel.js`

Назначение: Панель статистики (FPS, машины, режим)

Отображаемая информация:

- Режим игры (AR/TOUCH/GYRO)
- Tracking статус (для AR)
- Пауза
- Активные машины
- Машины в пуле
- Глобальный масштаб
- Радиус камеры (для TOUCH/GYRO)

Методы:

- `show()` - Показать панель
- `update(data)` - Обновить данные
- `hide()` - Скрыть панель

Возможности:

- Сворачивание/разворачивание (клик на заголовок)
- Компактный режим для мобильных
- Иконка ▼ / ▲ для индикации состояния

`src/ui/ControlPanel.js`

Назначение: Панель управления машинками

Функциональность для каждой модели:

- **Масштаб:** Слайдер 0.1x - 10x
- **Количество:** Слайдер 0 - 10 машин
- **Кнопки:**
 - - Добавить одну машину
 - - Удалить одну машину
- **Счетчик:** Текущее количество машин

Дополнительно:

- Кнопка "СБРОСИТЬ" - вернуть к начальным настройкам
- Отображение общего количества машин
- Сворачивание/разворачивание панели

Модели (по умолчанию):

1. Buggy - 3 шт, масштаб 1.0x
2. Milk Truck - 2 шт, масштаб 1.0x
3. Duck - 2 шт, масштаб 1.0x

Методы:

- `show()` / `hide()` - Показать/скрыть
- `toggle()` - Свернуть/развернуть
- `spawnSpecificModel(modelName)` - Добавить машину
- `removeSpecificModel(modelName)` - Удалить машину
- `setModelCount(modelName, count)` - Установить количество
- `updateModelScale(modelName, scale)` - Изменить масштаб
- `resetAll()` - Сбросить все настройки



Процесс работы приложения

Последовательность запуска:

1. index.html загружается
↓
2. main.js инициализируется
↓
3. StartScreen.js показывает стартовый экран
↓
4. Пользователь выбирает режим + настройки
↓
- 5a. AR режим:
 - ar_webxr.js
 - WebXR инициализация
 - Hit-test поиск поверхности
 - Размещение ковра
 - Калибровка
- 5b. TOUCH/GYRO режим:
 - nonAr.js
 - Three.js сцена
 - Орбитальная камера
 - Управление мышью/тироскопом
6. Загрузка моделей (CarModels.js)
↓
7. Создание дорожной сети (road_system.js)
↓
8. Спавн машин (TrafficManager.js)
↓
9. Render loop (обновление каждого кадра):
 - Обновление машин (Car.update)
 - Проверка коллизий
 - Респавн завершивших путь
 - Обновление UI (StatsPanel)
↓
10. Пользователь может:
 - Управлять камерой
 - Менять количество машин (ControlPanel)
 - Переключать режимы (ModeUI)

Исправленные баги

Баг #1: Зависание после подтверждения ковра (AR)

Проблема: После нажатия "ГОТОВО" рендер зависал

Причина: Hit-test продолжал выполняться после калибровки

Решение:

```
javascript

// ДО (неправильно):
if (hitTestSource) {
    // Hit-test выполнялся всегда
}

// ПОСЛЕ (правильно):
if (hitTestSource && !isCalibrated) {
    // Hit-test только ДО калибровки
}
```

Баг #2: Машины ехали боком после поворота

Проблема: Rotation не учитывал ориентацию модели

Решение:

```
javascript

// Плавная интерполяция угла
this.targetRotation = Math.atan2(dy, dx);
let rotDiff = this.targetRotation - this.currentRotation;

// Нормализация угла
while (rotDiff > Math.PI) rotDiff -= 2 * Math.PI;
while (rotDiff < -Math.PI) rotDiff += 2 * Math.PI;

// Плавное вращение
this.currentRotation += rotDiff * this.rotationSpeed;

// Применение с учетом ориентации модели
this.model.rotation.y = -this.currentRotation + Math.PI / 2;
```

Развёртывание

Локальный запуск:

```
bash

# Простой HTTP сервер (Python)
python server.py

# Или Python 3
python3 -m http.server 8000

# Или Node.js
npx http-server -p 8000
```

Требования:

- **AR режим:** HTTPS (WebXR требует защищенный контекст)
- **Браузеры:**
 - Chrome 90+ (Android/Desktop)
 - Safari 15+ (iOS)
 - Edge 90+

Production deploy:

1. Загрузить все файлы на HTTPS хостинг
2. Проверить пути к ассетам (модели, текстуры)
3. Убедиться что CDN ссылки работают (Three.js)

Поддерживаемые устройства

AR режим:

- Android (Chrome, Samsung Internet)
- iOS 15+ (Safari)
- Desktop (нет AR камеры)

TOUCH режим:

- Все устройства с тачскрином
- Desktop с мышью

GYRO режим:

- Смартфоны с гироскопом
 - iOS (с запросом разрешения)
 - Desktop (нет гироскопа)
-

Будущие улучшения

1. Игровая механика:

- Счет очков
- Задания (доставь груз)
- Препятствия на дорогах

2. Графика:

- Анимации машин (колеса, сигналы)
- Частицы (пыль, дым)
- Тени от машин

3. Звук:

- Звуки моторов
- Гудки
- Фоновая музыка

4. Мультиплеер:

- WebRTC для синхронизации
- Общая дорога для нескольких игроков

5. Редактор дорог:

- Создание своих карт
- Сохранение/загрузка

Контакты и поддержка

GitHub: <https://github.com/den812/ar-carpet-game>

Разработчик: ДЕВ-ЛИД фронтенд команды

Версия: V21 (Декабрь 2024)

Лицензия

Проект использует:

- Three.js (MIT License)
 - GLB модели (публичные/собственные)
 - Текстуры ковра (собственные)
-

Документация создана 27.12.2024