

Документация проекта AR Carpet Game

Обзор проекта

AR Carpet Game - интерактивная игра с машинками на детском ковре с дорогами. Поддерживает три режима работы:

- **AR** (дополненная реальность) - через WebXR API
- **TOUCH** (сенсорное управление) - орбитальная камера
- **GYRO** (гироскоп) - управление наклоном устройства

Текущая версия: V27 (Январь 2026)

Основные возможности:

- Реалистичное движение машин по дорогам
- Двустороннее движение с правосторонним трафиком
- Система коллизий между машинами
- 228 узлов дорожной сети с круговыми развязками
- Управление машинами в реальном времени
- Детальная статистика и отладка
- On-screen логирование для мобильных устройств

Структура проекта

```
ar-carpet-game/
├── index.html      # Главная страница V26
└── assets/
    ├── carpet-scan.jpg  # Текстура ковра (обновлена)
    └── models/          # 3D модели машин
        ├── Buggy.glb
        ├── CesiumMilkTruck.glb
        └── Duck.glb
└── src/
    ├── main.js         # V27 - Точка входа с диагностикой
    └── config.js       # Конфигурация
```

```
└── ar_webxr.js      # V26 - AR режим (WebXR)
└── nonAr.js        # V24 - TOUCH/GYRO режимы
└── chrome_diagnostic.js # Диагностика Chrome Android
└── cars/           # Логика машин
    └── Car.js       # V23 - Класс машины
    └── CarModels.js # Загрузчик моделей
└── roads/          # Дорожная система
    └── roadNetwork.js # V23 - Граф дорог
    └── road_system.js # V24 - Структура дорог
└── traffic/         # Управление трафиком
    └── traffic_manager.js # V23
└── ui/              # Пользовательский интерфейс
    └── StartScreen.js # V26 - Стартовый экран
    └── ModeUI.js     # Переключатель режимов
    └── StatsPanel.js # V3 - Статистика
    └── ControlPanel.js # Управление машинками
    └── OnScreenLogger.js # V2 - NEW! Логи на экране
```

☰ Детальное описание модулей

🏠 Корневые файлы

index.html V26

Назначение: Главная HTML страница приложения

Import Map:

```
javascript

{
  "three": "https://cdn.jsdelivr.net/npm/three@0.158.0/",
  "three/addons": "https://cdn.jsdelivr.net/npm/three@0.158.0/examples/jsm/",
  "mindar-image-three": "https://cdn.jsdelivr.net/npm/mind-ar@1.2.5/"
}
```

Основные элементы:

- `#start` - Стартовый экран с выбором режима
- `#ar-container` - Контейнер для AR сессии
- `#mode-ui` - Переключатель режимов (скрыт по умолчанию)

Настройки на стартовом экране:

1. Показать статистику (включено по умолчанию)
2. Панель управления машинками (включено по умолчанию)
3. Показать логи (отладка) - NEW!
4. Инверсия управления
5. Показать дороги (отладка)

Стили:

- Адаптивный дизайн (мобильные + десктоп)
 - Неоновый стиль кнопок (зеленый градиент с тенью)
 - Темная тема с полупрозрачными элементами
 - Фиксированное позиционирование UI (z-index: 1000-10000)
-

Основные модули

src/main.js V27

Назначение: Точка входа приложения, управление режимами, диагностика

Глобальные объекты:

```
javascript

const logger = new OnScreenLogger(); // Логгер на экране
window.logger = logger; // Доступен глобально для отладки
let currentMode = null; // Текущий режим (AR/TOUCH/GYRO)
```

Основные функции:

diagnoseEnvironment()

Диагностика окружения браузера.

Возвращает объект:

```
javascript
```

```
{  
  userAgent: string,  
  isChrome: boolean,  
  isSamsung: boolean,  
  isEdge: boolean,  
  hasWebXR: boolean,  
  isHTTPS: boolean,  
  screenWidth: number,  
  screenHeight: number,  
  pixelRatio: number  
}
```

checkSystemReady()

Проверка готовности всех модулей перед запуском.

Проверяет:

- Three.js загружен
- Все модули доступны (roads, traffic, cars)
- Нет ошибок импорта

Возвращает: `Promise<boolean>`

run(mode, settings)

Основная функция запуска выбранного режима.

Параметры:

- `mode` - 'AR' | 'TOUCH' | 'GYRO'
- `settings` - объект настроек:

javascript

```
{  
  showStats: boolean,  
  showControl: boolean,  
  showLogger: boolean, // NEW!  
  invertControls: boolean,  
  showRoads: boolean  
}
```

Логика работы:

1. Диагностика окружения
2. Управление логгером (если `(showLogger === true)`)
3. Специальная обработка для Chrome Android
4. Проверка готовности системы
5. Запуск соответствующего режима
6. Обработка ошибок с автоматическим fallback на TOUCH

Особенности для Chrome Android:

- Дополнительная задержка 500мс для инициализации
- Детальная проверка модулей
- Graceful degradation при ошибках

`changeMode(mode)`

Переключение между режимами с перезагрузкой страницы.

Параметры:

- `mode` - новый режим

Действия:

- Проверка что режим отличается от текущего
- Загрузка настроек из localStorage
- Полная перезагрузка страницы (`(location.reload())`)

`src/config.js`

Назначение: Централизованная конфигурация игры

Основные параметры:

```
javascript
```

```

export const CONFIG = {
  carScales: {
    defaultScale: 0.002,
    models: {
      "Buggy.glb": 0.3, // Маленькая гоночная машина
      "Duck.glb": 10.0, // Игрушечная утка
      "CesiumMilkTruck.glb": 8.5 // Грузовик
    }
  },
  carpet: {
    width: 2.0, // Ширина ковра в метрах
    height: 2.5, // Высота ковра в метрах
    y: 0 // Высота над землей
  },
  camera: {
    fov: 50,
    near: 0.01,
    far: 100,
    position: { x: 0, y: 1.5, z: 1.8 }
  },
  cars: {
    count: 7, // Количество машин по умолчанию
    baseSpeed: 0.0005, // Базовая скорость
    speedVariation: 0.0003, // Разброс скорости
    heightAboveCarpet: 0.0 // На уровне ковра
  },
  roads: {
    laneOffset: 0.02 // Смещение полосы от центра (1/4 ширины)
  }
}

```

Экспортируемые функции:

`getCarScale(modelName)`

Возвращает итоговый масштаб для модели.

Параметры: `modelName` - имя файла модели

Возвращает: `number` (`defaultScale * multiplier`)

Пример:

javascript

```
getCarScale("Buggy.glb") // 0.002 * 0.3 = 0.0006
```

```
updateConfig(path, value)
```

Обновляет параметр конфигурации.

Параметры:

- `[path]` - строка пути к параметру ('cars.count')
- `[value]` - новое значение

Пример:

```
javascript
```

```
updateConfig('cars.count', 10);
```

🚗 Модуль машин

```
src/cars/Car.js V23
```

Назначение: Класс отдельной машины с логикой движения

Конструктор:

```
javascript
```

```
constructor(model, roadNetwork, modelName = 'unknown')
```

Свойства:

```
javascript
```

```
{
  model: THREE.Group,      // 3D модель
  roadNetwork: RoadNetwork, // Ссылка на дорожную сеть
  modelName: string,       // Имя модели
  baseSpeed: number,        // Базовая скорость (+ случайный разброс)
  currentSpeed: number,     // Текущая скорость
  heightAboveRoad: number,  // Высота над дорогой (0.0)
  path: Array<Node>,       // Массив узлов маршрута
  currentPathIndex: number, // Текущий сегмент пути
  progress: number,         // Прогресс на сегменте (0-1)
  currentLane: Lane,        // Текущая полоса движения
  targetRotation: number,   // Целевой угол поворота
  currentRotation: number,  // Текущий угол поворота
  rotationSpeed: number,    // Скорость поворота (0.15)
  isActive: boolean,        // Машина активна
  isStopped: boolean        // Машина остановлена (коллизия)
}
```

Методы:

`spawn(startNode, endNode)`

Размещает машину на дороге и запускает движение.

Параметры:

- `startNode` - начальный узел
- `endNode` - конечный узел

Возвращает: `boolean` - успешность спавна

Действия:

1. Проверка валидности узлов
2. Построение пути через A* алгоритм
3. Установка начальной позиции с учетом полосы
4. Установка начальной ориентации
5. Активация машины

Особенности:

- Проверка что `startNode ≠ endNode`

- Минимальная длина пути: 2 узла
- Автоматическое смещение на правую полосу

update()

Обновление позиции и поворота машины (вызывается каждый кадр).

Логика работы:

1. Проверка валидности текущего и следующего узла
2. Проверка остановки (коллизия)
3. Вычисление адаптивной скорости:
 - Замедление перед поворотами
 - Замедление при приближении к следующему узлу
4. Обновление прогресса на сегменте
5. Переход на следующий сегмент при progress >= 1
6. Интерполяция позиции с учетом смещения полосы
7. Плавный поворот к целевому углу

Адаптивная скорость:

javascript

```
const turnFactor = turnAngle > 0.5 ? 0.6 : 1.0;
const distanceFactor = distanceToTurn < 0.2 ? 0.7 : 1.0;
this.currentSpeed = this.baseSpeed * turnFactor * distanceFactor;
```

Плавный поворот:

javascript

```

// Нормализация угла [-π, π]
let rotDiff = this.targetRotation - this.currentRotation;
while (rotDiff > Math.PI) rotDiff -= 2 * Math.PI;
while (rotDiff < -Math.PI) rotDiff += 2 * Math.PI;

// Плавная интерполяция
this.currentRotation += rotDiff * this.rotationSpeed;

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

```

`despawn()`

Удаляет машину с дороги.

Действия:

- Деактивация (`(isActive = false)`)
- Скрытие модели (`(visible = false)`)
- Очистка пути и состояния

`setGlobalScale(scale)`

Применяет глобальный масштаб к машине.

Параметры: `scale` - множитель масштаба

Формула: `finalScale = baseScale * scale`

`stopForCollision() / resumeMovement()`

Управление остановкой при коллизиях.

- `stopForCollision()` - останавливает машину (`(isStopped = true)`)
- `resumeMovement()` - возобновляет движение (`(isStopped = false)`)

`checkCollision(otherCar)`

Проверяет коллизию с другой машиной.

Параметры: `otherCar` - другой экземпляр Car

Возвращает: `boolean`

Логика: Проверка расстояния между машинами (минимум: 0.15)

`applyRandomColor()`

Применяет случайный цвет к машине.

Логика: HSL цвет с случайным Hue, фиксированными Saturation (0.7) и Lightness (0.5)

`smoothstep(t)`

Плавная интерполяция для движения.

Формула: $t * t * (3 - 2 * t)$

`src/cars/CarModels.js`

Назначение: Загрузчик и менеджер 3D моделей машин

Конструктор:

```
javascript

constructor() {
    this.loader = new GLTFLoader();
    this.models = [];
    this.isLoaded = false;
    this.modelList = [
        { name: 'Buggy.glb', path: './assets/models/Buggy.glb' },
        { name: 'CesiumMilkTruck.glb', path: './assets/models/CesiumMilkTruck.glb' },
        { name: 'Duck.glb', path: './assets/models/Duck.glb' }
    ];
}
```

Методы:

`loadAll()`

Асинхронная загрузка всех моделей.

Возвращает: `Promise<void>`

Логика:

1. Проверка что модели еще не загружены
2. Параллельная загрузка через `Promise.all()`

3. Фильтрация успешно загруженных

4. Установка флага `isLoaded = true`

`loadModel(path, name)`

Загрузка одной модели.

Параметры:

- `path` - путь к файлу GLB
- `name` - имя модели

Возвращает: `Promise<{name, model}>` или `null` при ошибке

Особенности: Не останавливает загрузку других моделей при ошибке

`getRandomModel()`

Возвращает случайную модель.

Возвращает: `{name, model: cloned}`

Особенности: Клонирует модель для повторного использования

`getModelByName(name)`

Возвращает модель по имени.

Параметры: `name` - имя файла модели

Возвращает: `{name, model: cloned}` или `null`

Особенности: Всегда клонирует модель

Дорожная система

`src/roads/roadNetwork.js` V23

Назначение: Граф дорожной сети с валидацией

Конструктор:

javascript

```
constructor() {
    this.nodes = []; // Массив узлов
    this.roads = []; // Массив дорог
    this.lanes = []; // Массив полос движения
}
```

Структура узла:

```
javascript

{
    x: number,      // Координата X
    y: number,      // Координата Y
    connections: Node[] // Массив соседних узлов
}
```

Структура дороги:

```
javascript

{
    start: Node, // Начальный узел
    end: Node, // Конечный узел
    length: number // Длина дороги
}
```

Структура полосы:

```
javascript

{
    start: Node,      // Начальный узел
    end: Node,        // Конечный узел
    direction: {x, y}, // Вектор направления
    offset: {x, y},   // Смещение от центра
    length: number    // Длина полосы
}
```

Методы:

```
addNode(x, y)
```

Добавляет узел в сеть с проверкой валидности.

Параметры: координаты `x`, `y`

Возвращает: `Node` или `null`

Валидация:

- Проверка типов (`typeof === 'number'`)
- Проверка NaN
- Проверка дубликатов (расстояние < 0.01)

`addRoad(startNode, endNode)`

Создает двустороннюю дорогу с двумя полосами.

Параметры: начальный и конечный узлы

Возвращает: `Road` или `null`

Логика:

1. Проверка валидности узлов
2. Проверка дубликатов дороги
3. Вычисление длины дороги
4. Создание 2 полос (по одной в каждом направлении):
 - **Полоса 1:** `start → end` (смещение вправо)
 - **Полоса 2:** `end → start` (смещение вправо)
5. Добавление двунаправленных связей между узлами

Смещение полос:

javascript

```
const laneOffset = 0.02; // 1/4 ширины дороги (0.08)
const angle = Math.atan2(end.y - start.y, end.x - start.x);
const perpAngle = angle + Math.PI / 2;
const offsetX = Math.cos(perpAngle) * laneOffset;
const offsetY = Math.sin(perpAngle) * laneOffset;
```

`findPath(start, end, maxDepth = 20)`

A* поиск пути между узлами с валидацией.

Параметры:

- `start` - начальный узел
- `end` - конечный узел
- `maxDepth` - максимальная длина пути

Возвращает: `Array<Node>` или `[]`

Логика A*:

1. Создание открытого и закрытого множества
2. Сортировка по стоимости ($\text{cost} + \text{heuristic}$)
3. Поиск кратчайшего пути
4. Валидация найденного пути

Heuristic: Евклидово расстояние между узлами

Fallback стратегии:

- Если путь не найден → случайный сосед
- Если нет соседей → случайный узел из сети
- Валидация всех узлов в пути

`validatePath(path)`

Проверяет валидность пути.

Параметры: `path` - массив узлов

Возвращает: `boolean`

Проверки:

- Минимум 2 узла
- Все узлы определены
- Все координаты валидны

`getLane(fromNode, toNode)`

Возвращает полосу движения между узлами.

Параметры: начальный и конечный узлы

Возвращает: `Lane` или `null`

`getStats()`

Статистика дорожной сети.

Возвращает:

```
javascript
{
  nodes: number,      // Количество узлов
  roads: number,      // Количество дорог
  lanes: number,      // Количество полос
  avgConnections: string // Среднее число связей на узел
}
```

`src/roads/road_system.js` V24

Назначение: Создание структуры дорог конкретного ковра

Функция `createRoadNetwork(parent, options)`:

Параметры:

- `parent` - THREE.Group для визуализации
- `options` - объект опций:

```
javascript
{
  showRoads: boolean // Показать визуализацию дорог
}
```

Возвращает: `RoadNetwork`

Структура дорог:

228 узлов распределенных по ковру, включая:

1. **3 круговые развязки:**

- Верхняя левая: `(-0.32, 0.67)`, радиус 0.08

- Средняя левая: `(-0.21, -0.06)`, радиус 0.08
- Нижняя левая: `(-0.04, -0.88)`, радиус 0.08

2. Основная кольцевая дорога (228 последовательных узлов)

3. Синие соединения (перекрестки):

- От развязок к центру: `[19→28, 24→29, 28→32]`
- Центральные: `[36→44, 40→108, 44→45]`
- Внутренние: `[32→166, 33→167, 34→168, 35→157]`
- Правая сторона: `[73→94, 94→113, ...]`
- И другие перекрестки

Валидация узлов:

```
javascript

// Проверка каждого узла перед добавлением
if (typeof node.x !== 'number' || typeof node.y !== 'number' ||
    isNaN(node.x) || isNaN(node.y) ||
    !isFinite(node.x) || !isFinite(node.y)) {
  console.error(`❌ Невалидный узел #${i}:`, node);
  continue;
}
```

Статистика:

- Валидные узлы: ~228
- Валидные дороги: ~260+
- Полосы движения: ~520+ (по 2 на дорогу)

Визуализация (если `showRoads = true`):

1. Дорожные сегменты:

- Серые плоскости (0x333333)
- Ширина: 0.08
- Высота: 0.001 (над ковром)

2. Центральные линии:

- Белые пунктирные (0xffffffff)

- DashedLine (dashSize: 0.03, gapSize: 0.02)

3. Круговые развязки:

- Серые кольца (дорога)
- Зеленые центры (0x4a7c4e)
- Желтые точки в центре (0xfffff00)

Try-catch блоки:

- Создание дорожной сети
 - Визуализация (не прерывает работу при ошибке)
-

Управление трафиком

[src/traffic/traffic_manager.js] V23

Назначение: Спавн, обновление и управление всеми машинами

Конструктор:

```
javascript

constructor(parent, roadNetwork) {
    this.parent = parent;          // THREE.Group
    this.roadNetwork = roadNetwork; // RoadNetwork
    this.cars = [];                // Массив всех машин
    this.carPool = [];              // Пул (не используется)
    this.globalScaleMultiplier = 1.0; // Глобальный масштаб
    this.isInitialized = false;     // Флаг инициализации
    this.carModels = null;          // CarModels instance
}
```

Методы:

[init()]

Инициализация менеджера и загрузка моделей.

Возвращает: **[Promise<void>]**

Действия:

1. Проверка что уже не инициализирован

2. Валидация дорожной сети (минимум 2 узла)
3. Загрузка статистики сети
4. Создание и инициализация CarModels
5. Установка флага `isInitialized = true`

`spawnCars(count)`

Спавнит заданное количество машин разных типов.

Параметры: `count` - количество машин

Возвращает: `Promise<void>`

Распределение по умолчанию:

- 3x Buggy.glb
- 2x CesiumMilkTruck.glb
- 2x Duck.glb

Логика:

1. Проверка инициализации
2. Последовательный спавн машин по типам
3. Задержка 100мс между спавнами
4. Логирование успешно заспавненных машин

`spawnCarWithModel(modelData)`

Спавнит одну машину конкретной модели.

Параметры: `modelData` - `{name, model}`

Возвращает: `Promise<Car | null>`

Логика:

1. Валидация `modelData`
2. Получение случайных валидных узлов
3. Проверка что узлы различаются
4. Тестовое построение пути

5. Создание экземпляра Car
6. Добавление модели в сцену
7. Применение глобального масштаба
8. Вызов `car.spawn(startNode, endNode)`
9. Проверка успешности спавна
10. Возврат машины или null

Валидация узлов:

```
javascript

// Проверка валидности координат
if (!endNode || endNode === startNode ||
    typeof endNode.x !== 'number' ||
    typeof endNode.y !== 'number') {
  console.error('❌ Не удалось найти валидный конечный узел');
  return null;
}
```

`update()`

Обновление всех машин (вызывается каждый кадр).

Логика:

1. Проверка инициализации
2. Фильтрация активных машин
3. **Система коллизий:**
 - Двойной цикл по активным машинам
 - Проверка расстояния между парами
 - Остановка обеих машин при коллизии
 - Возобновление движения если нет коллизий
4. **Обновление машин:**
 - Вызов `car.update()` для каждой
 - Try-catch для обработки ошибок
 - Респавн завершивших путь (через 0.5-2.5 сек)

Система коллизий:

javascript

```
for (let i = 0; i < activeCars.length; i++) {
    const car1 = activeCars[i];
    let hasCollision = false;

    for (let j = i + 1; j < activeCars.length; j++) {
        const car2 = activeCars[j];

        if (car1.checkCollision(car2)) {
            hasCollision = true;
            car1.stopForCollision();
            car2.stopForCollision();
        }
    }

    if (!hasCollision) {
        car1.resumeMovement();
    }
}
```

`setGlobalScale(scale)`

Устанавливает глобальный масштаб для всех машин.

Параметры: `scale` - множитель масштаба

Действия:

- Сохранение в `globalScaleMultiplier`
- Применение ко всем существующим машинам
- Логирование