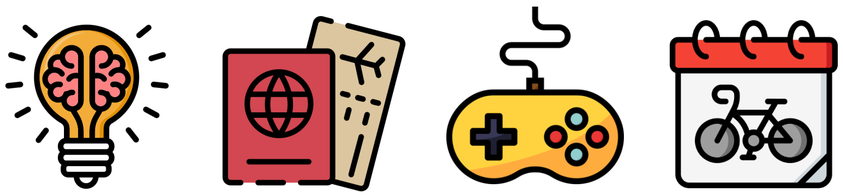
Как переселить целый город.

Смена игрового движка ЮVillage

Александр Непомнящих
Frontend-разработчик

Александр Непомнящих

Frontend-разработчик
ЮMoney



3 года геймификации

2 игры

1 выступление

Геймификация

Применение игровых механик для неигровых процессов с целью привлечения пользователей, повышения их вовлечённости в использовании продуктов и услуг

IOVillage

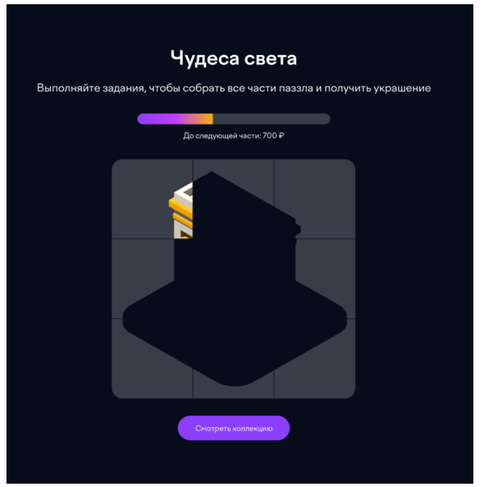
ЮVillage

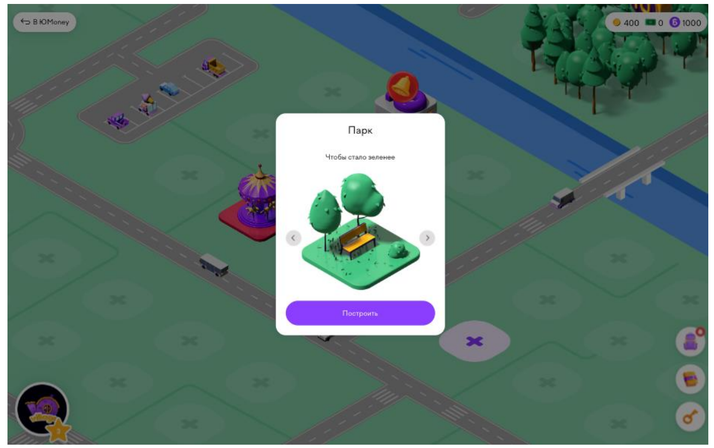
Запуск: 26 июля 2021


В ЮMoney
6440
155
132
15
village
13

Как играть в ЮVillage

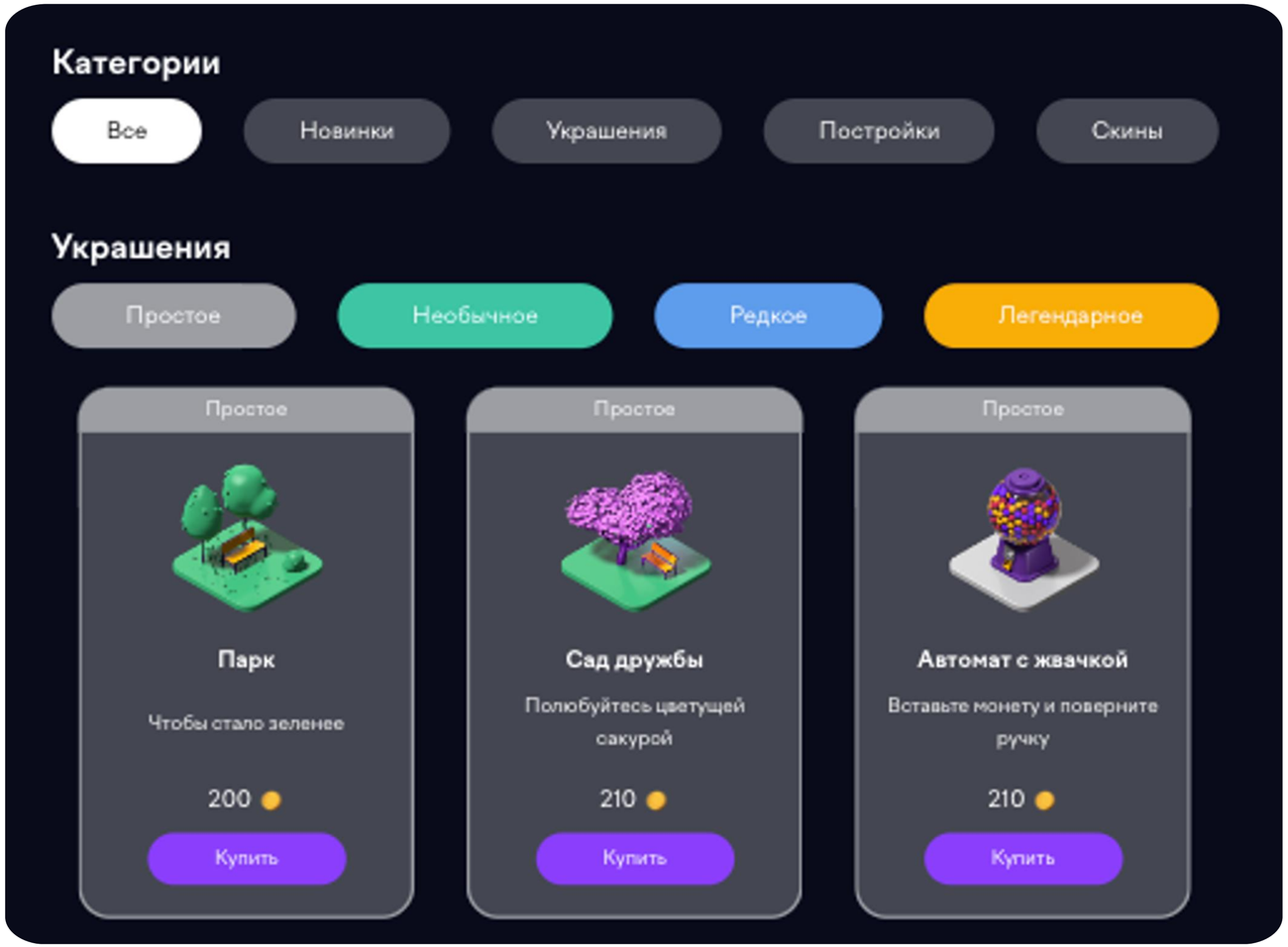
- Возводить постройки и украшать город
- Играть в мини-игры
- Собирать коллекции стикеров и паззлов
- Проходить тесты — например, «Какое твоё тотемное животное»





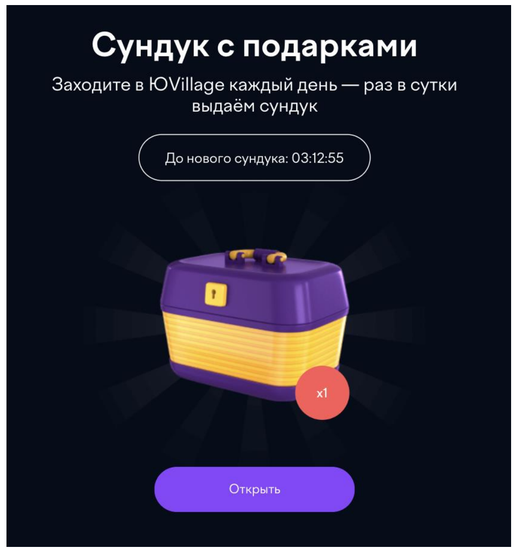
Где взять постройки и украшения

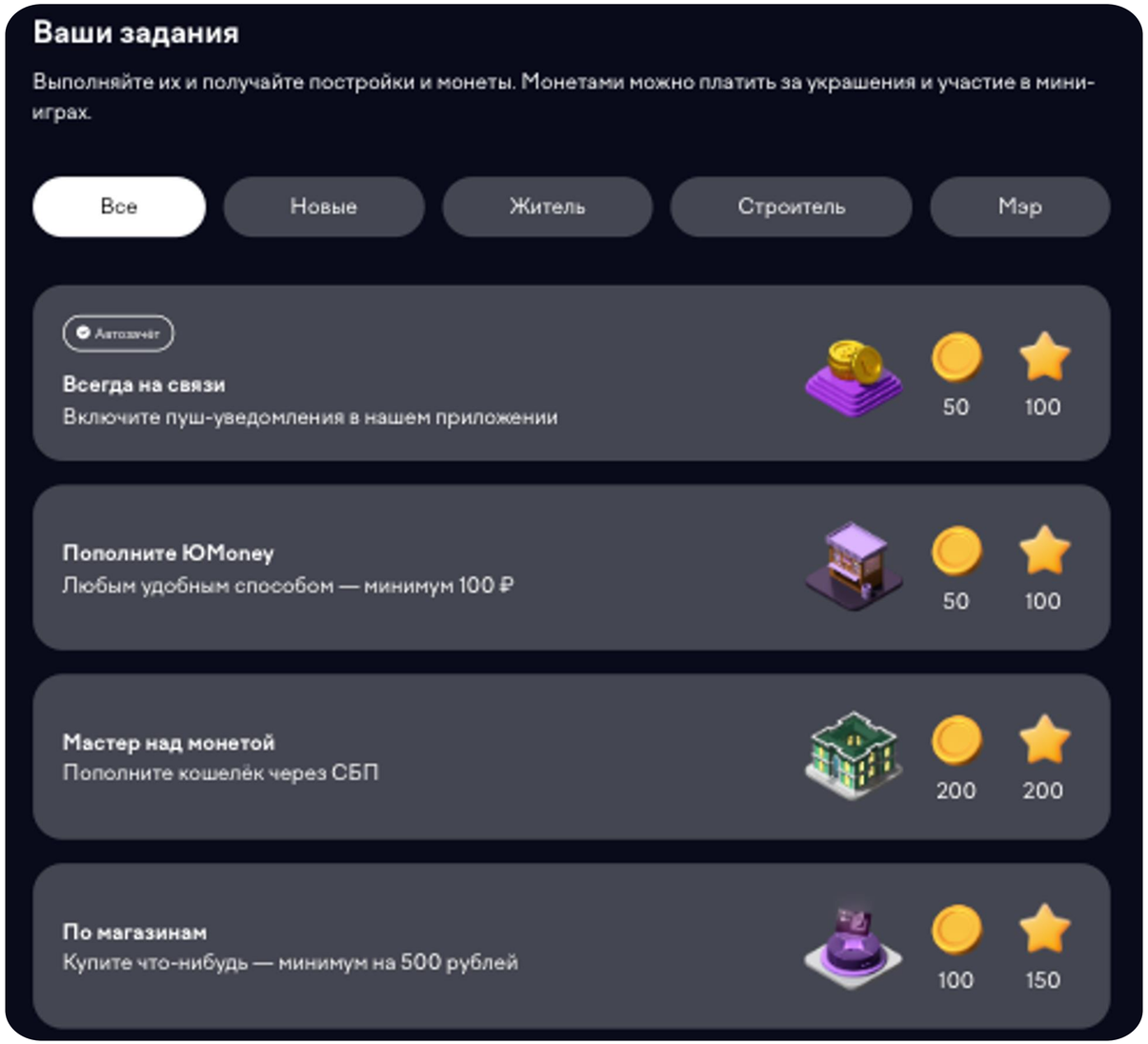
- Купить в магазине
- Получить за задания



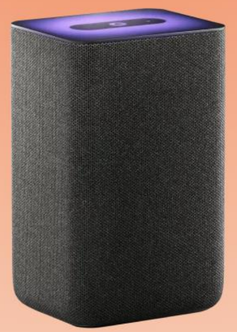
Как получить монеты

- Выполнять задания
- Ежедневно открывать сундуки
- Получать их за повышение уровня, выполняя квесты





Что разыгрывали



Умную колонку Яндекс Станция 2 с Алисой

Участвуют игроки с 5 по 15 уровень

Часы Samsung Galaxy Watch 6

Получат шанс выиграть игроки с 10 по 15 уровень

Смартфон Samsung Galaxy S23 Ultra

Разыгрывается между игроками 15 уровня

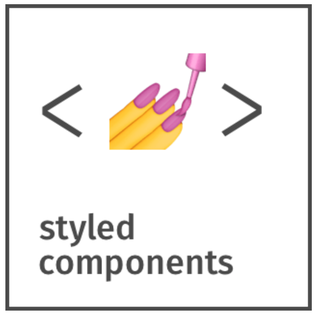
Первое решение

Рендеринг игровой карты: Canvas API

Отображение анимации:
- Window API
- Видео с прозрачным фоном

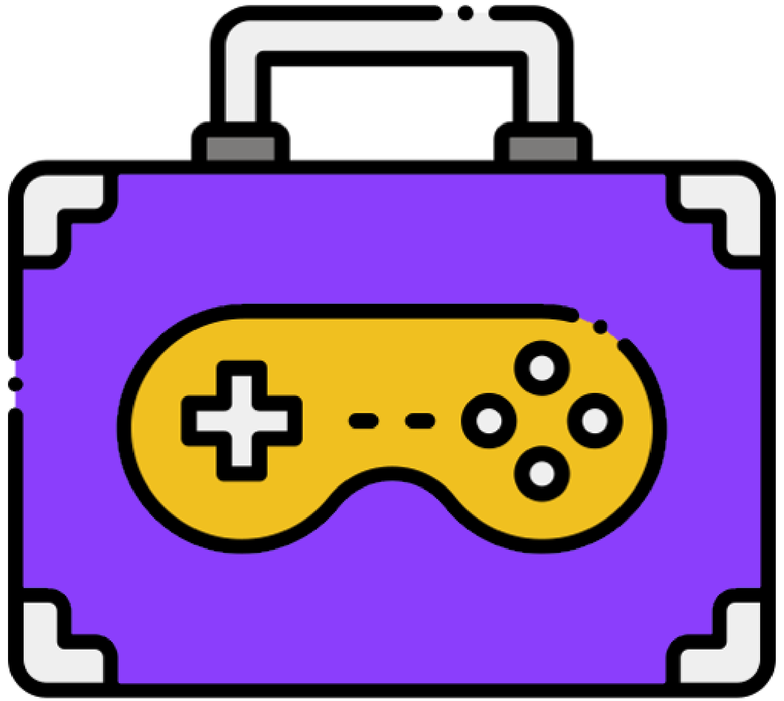
Игровой движок: собственный на JS

Интерфейс: React / styled-components



Почему не взяли готовый движок

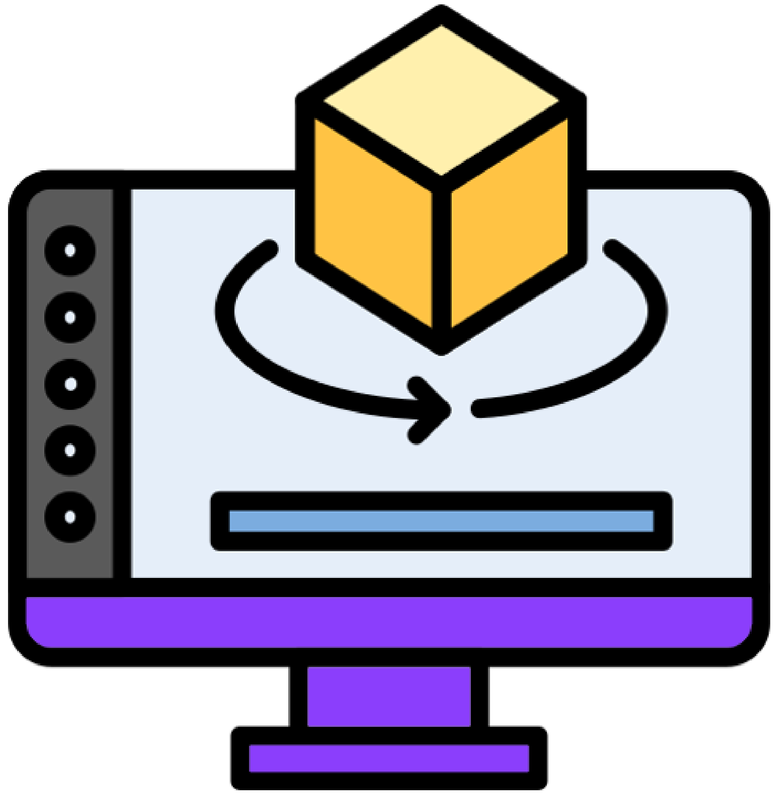
- Независимость от сторонних разработчиков
- Планировался небольшой функционал
- Сомневались в быстродействии
- Амбициозность

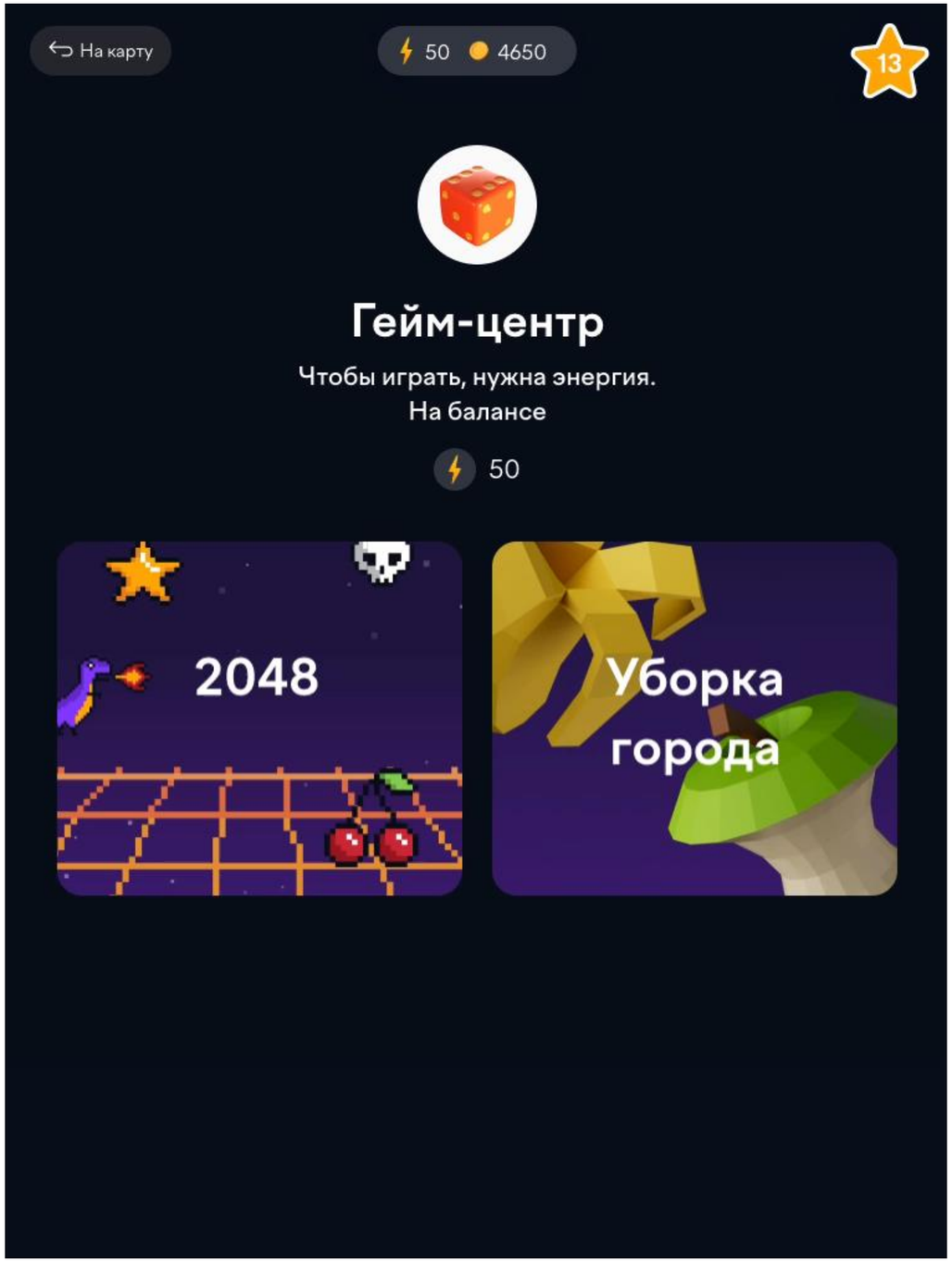


Мини-игры

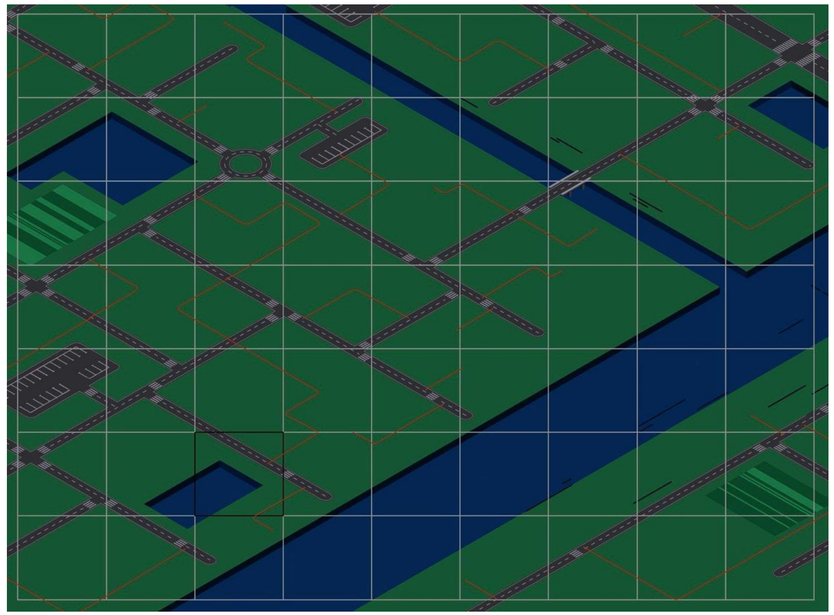
WebGL – это программная библиотека для JavaScript основанная на OpenGL, которая позволяет создавать 3D графику

ThreeJS – это библиотека JavaScript, содержащая набор готовых классов для создания и отображения интерактивной 3D графики в WebGL

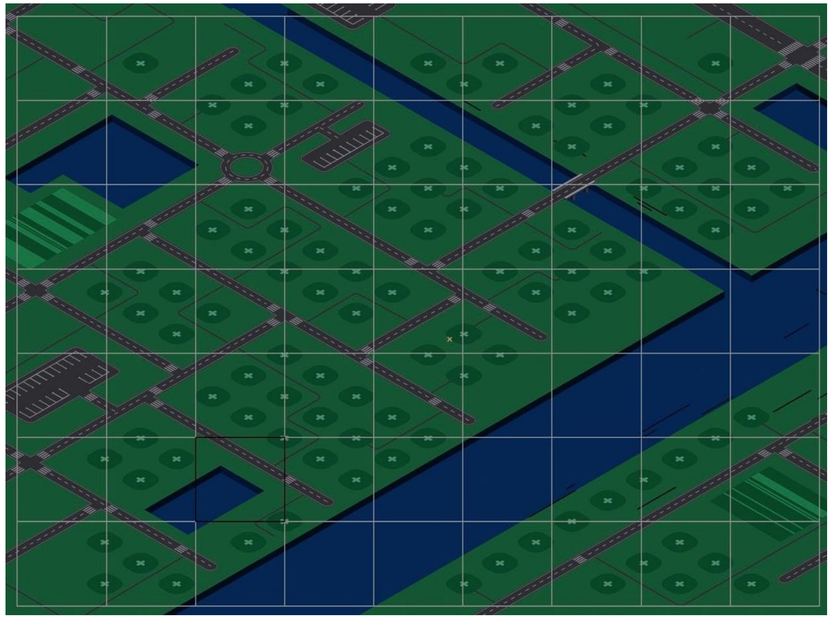



На карту
50
4650
13
Гейм-центр
Чтобы играть, нужна энергия.
На балансе
50
2048
Уборка
города

Как рисуем карту

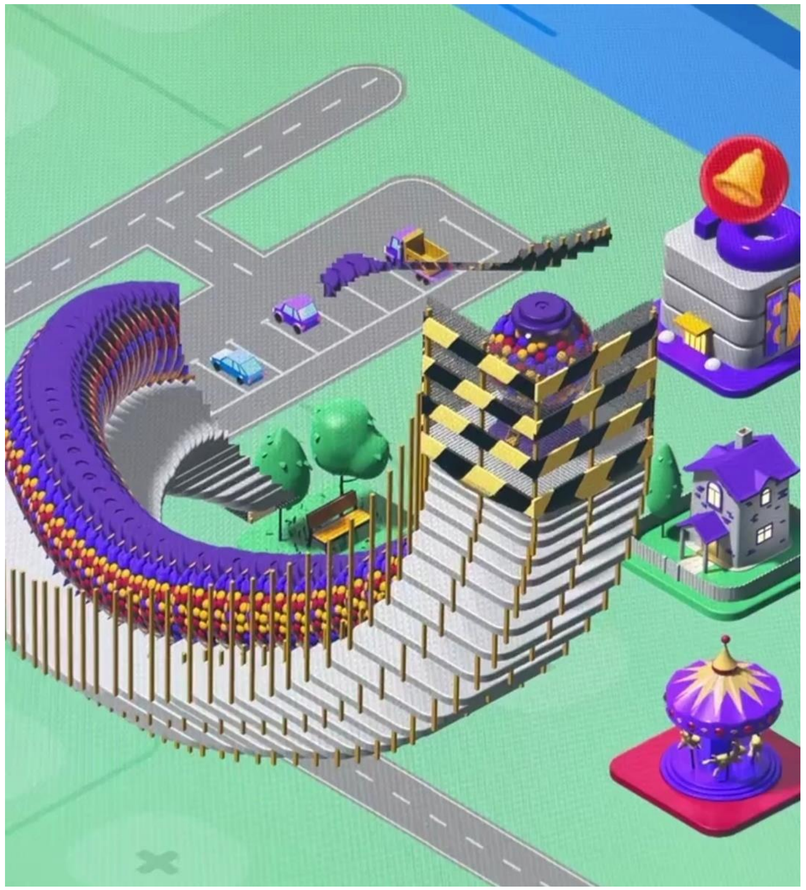


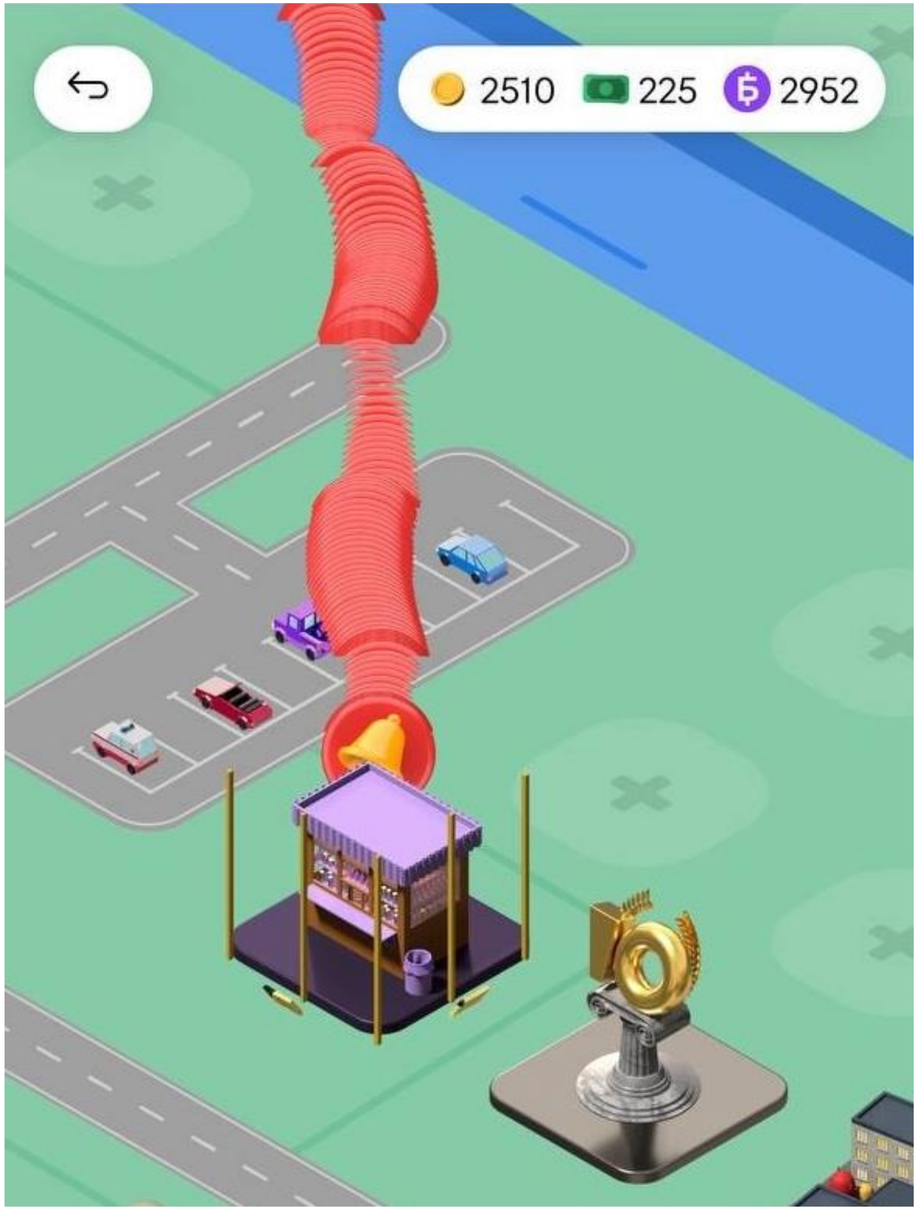
```
[
    [0,1,0,0,0,1,0,0,0,1,0,1,0,0,0,0,0,1,0,0],
    [0,0,0,0,1,0,1,0,0,0,1,0,0,0,1,0,0,0,0,0],
    [0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0],
    [0,0,0,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,0],
    [0,0,0,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0],
    [0,0,0,1,0,1,0,0,0,1,0,0,0,1,0,0,0,0,0,0],
    [0,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0],
    [0,1,0,0,0,1,0,1,0,0,0,1,0,1,0,1,0,0,0,0],
    [1,0,1,0,0,0,1,0,1,0,0,0,1,0,1,0,0,0,0,0],
    [0,1,0,1,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0],
    [0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0],
    [0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0],
    [0,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0],
    [0,1,0,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0],
    [1,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0],
    [0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
    [1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
    [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
    [1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
]
```



Сложности

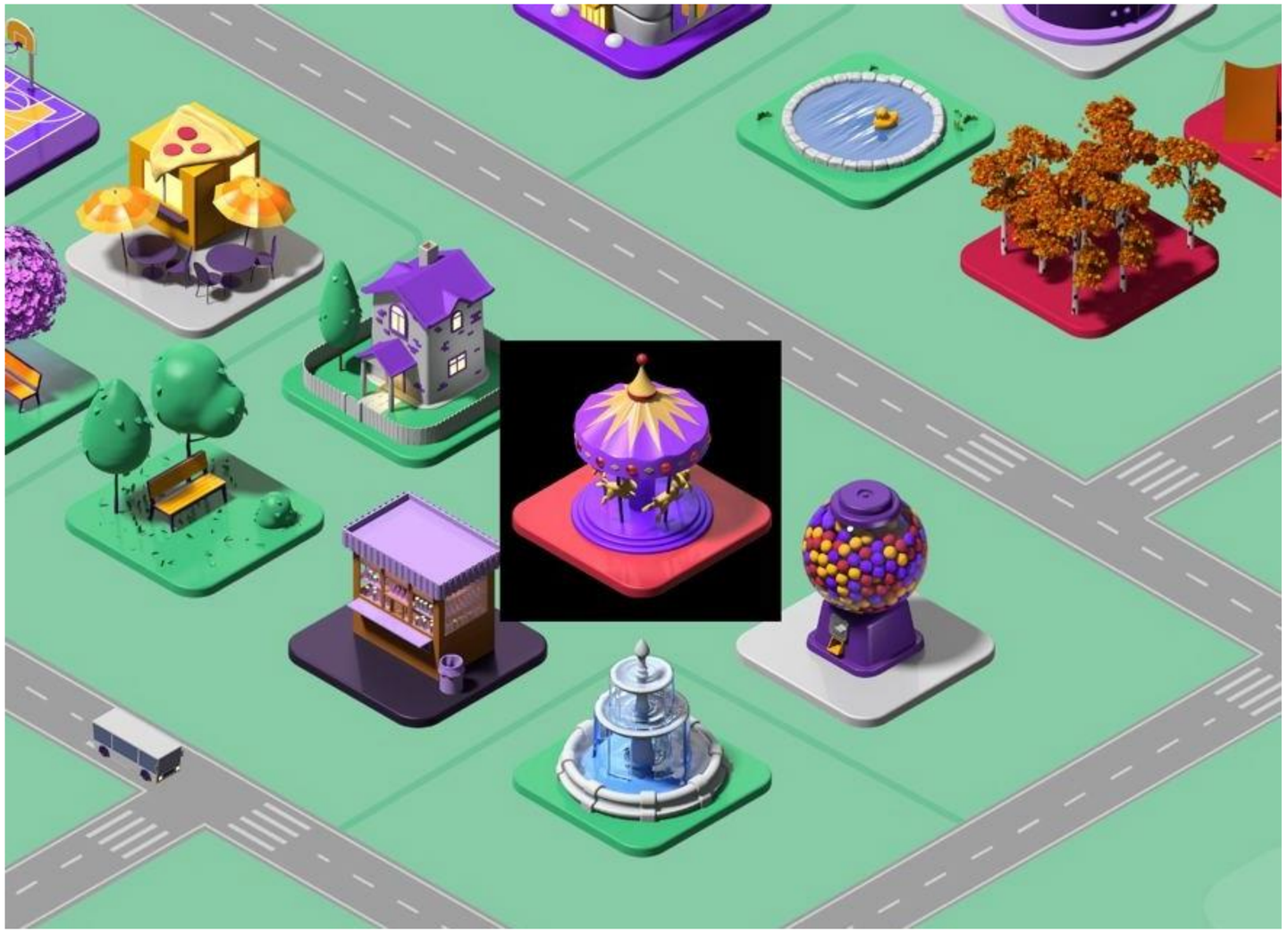
Анимации «глючные»





Сложности

- Видео анимация



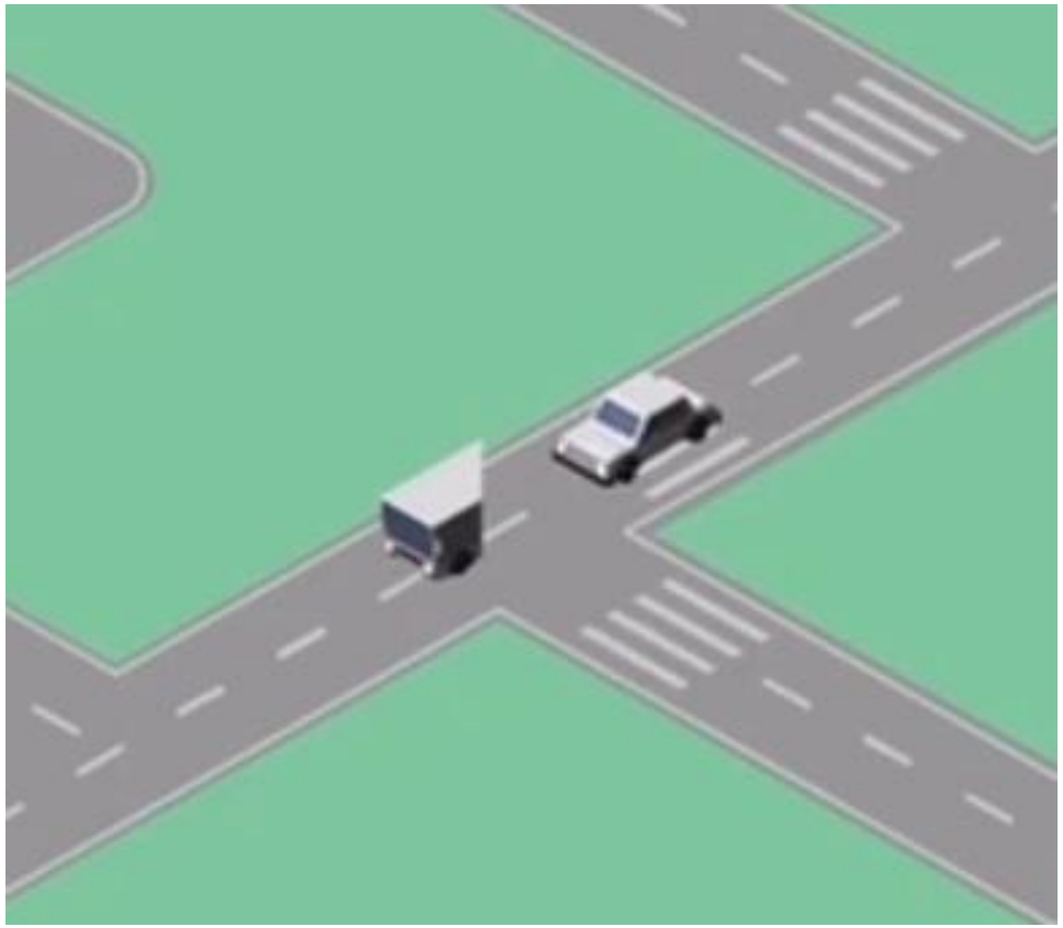
.mov в Safari / **.webM** в остальных браузерах

Сложности

- Сложность работы с слоями

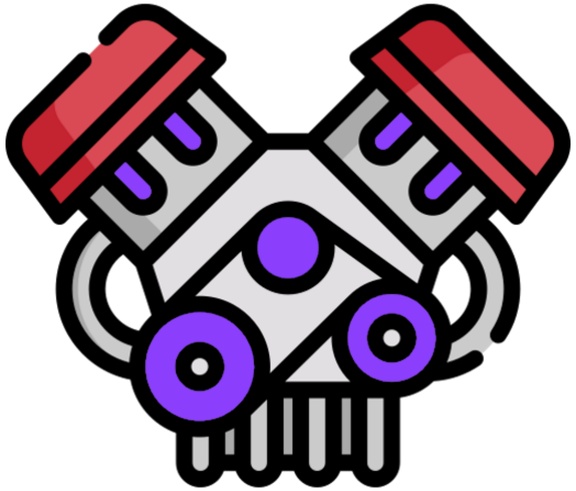
Сложности

- Нет физики объектов
- Низкая производительность
- Медленная навигация по карте
- Расширять карту сложно
- Трудоёмкость добавления нового функционала



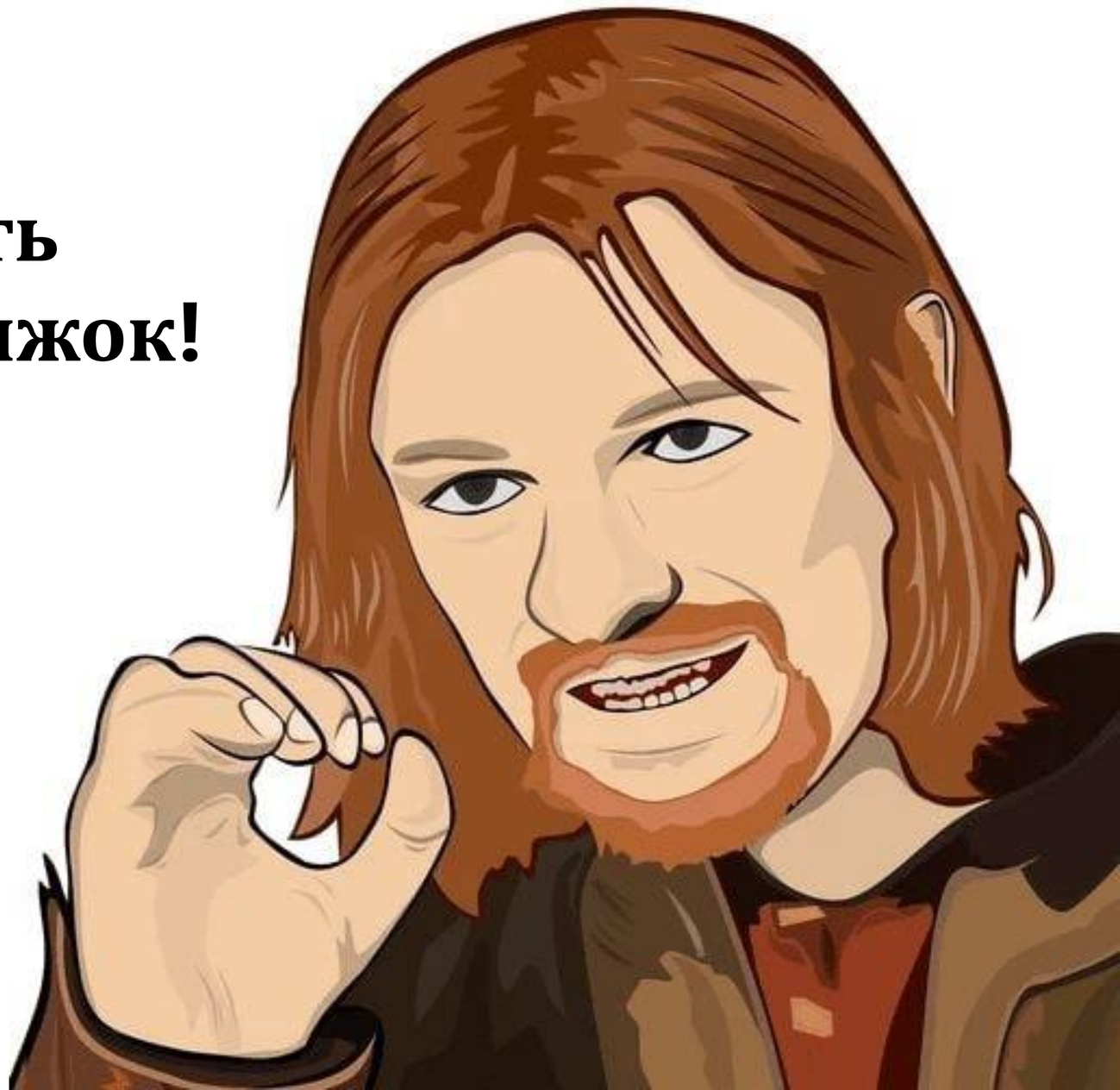
Новый игровой движок

Игровой движок



Набор инструментов, который позволяет работать с графикой, физикой, скриптами и другими элементами, с помощью которых можно создать игру

Нельзя просто так взять
и выбрать игровой движок!



Требования

- Встраиваемость в наш стек
- Возможность реализовать существующий функционал
- Простая работа с анимацией
- Производительность
- Портирование на мобилку
- Понятная документация
- Скорость и лёгкость разработки

Выборка

1. Gdevelop
2. Modd
3. Construct 3
4. ImpactJS
5. MelonJS
6. PixiJS
7. Phaser 3
8. Cocos 2D
9. BabylonJS
10. Isogenicengine
11. LimeJS

1. MelonJS
2. ImpactJS
3. Phaser 3
4. PixiJS
5. BabylonJS
6. Isogenicengine
7. LimeJS

Отсеяли:

Тяжёлая документация,
не нашли похожих
на наш проектов

Мало примеров
использования

Недостаточный
функционал

Заточен под 3D-игры,
высокий порог вхождения

Платный, не нашли
похожих проектов

PixiJS

Преимущества

- Бесплатная
- Высокопроизводительная
- Большое сообщество
- Низкий порог вхождения
- Понятная документация
- Поддержка TypeScript

Недостатки

- Нет встроенного портирования на мобилку
- Нет встроенной физики
- Новая 5 версия не поддерживает canvas

Преимущества

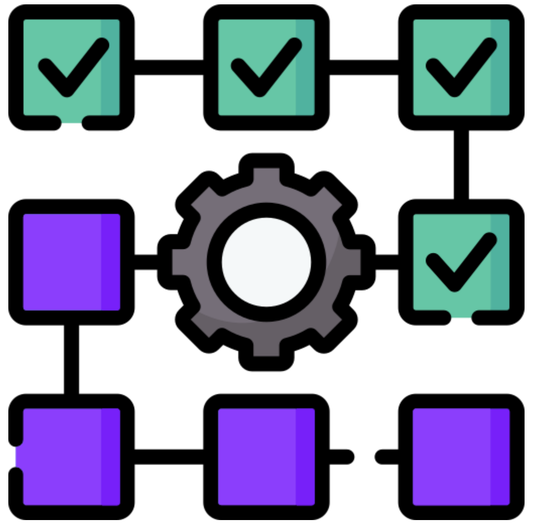
- Бесплатный
- Высокопроизводительный
- Поддержка TypeScript
- Богатый инструментарий
- Поддержка Canvas и WebGL
- Оптимизирован для браузеров
- Большое сообщество
- Низкий порог вхождения
- Понятная документация

Недостатки

- Нет встроенного портирования на мобилку

Тест-кейс

- Создание карты
- Отображение на карте строительных ячеек и зданий
- Клик по ячейкам и зданиям, добавление эффектов
- Интеграция новой карты в компонент игры
- Подключение к старому стейту
- Настройка спрайтовой анимации
- Настройка камеры
- Тест работы на мобилке и десктопе

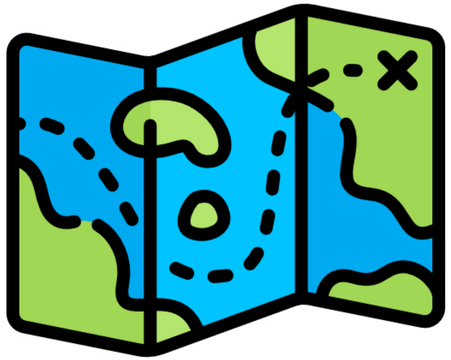


Результат: 100%

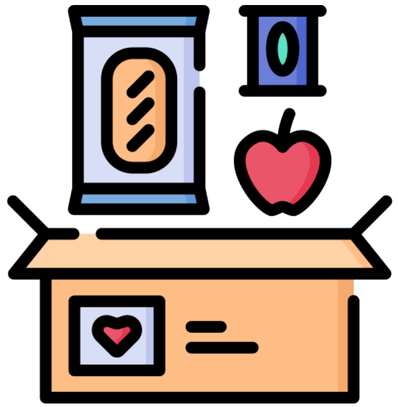
Как работать с Phaser JS

Что нужно для работы

Phaser

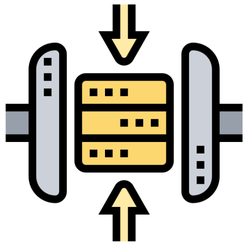


Tiled



Atlas-packer

Оптимизация



Сжатие изображений

Загрузка только необходимых изображений

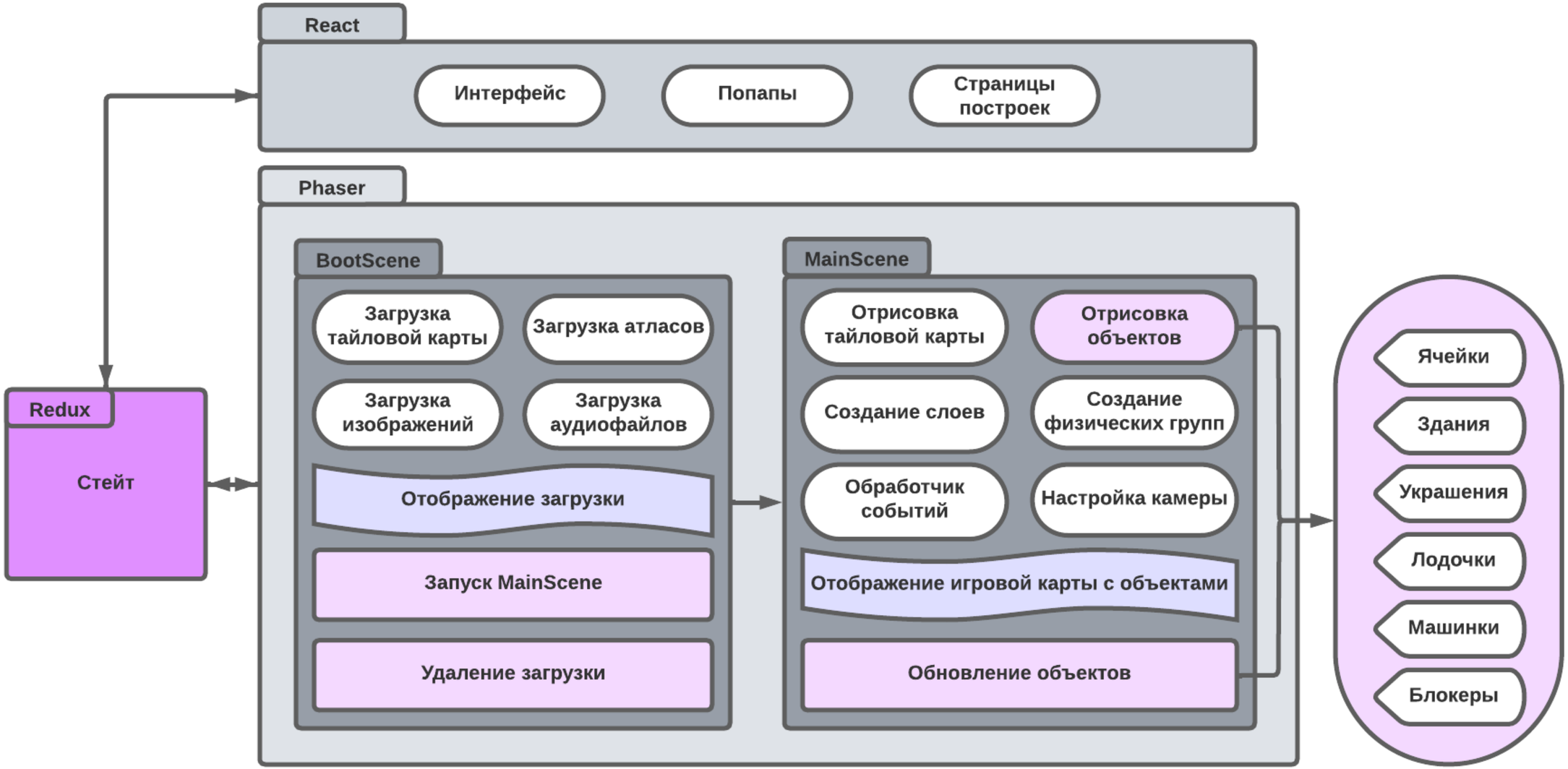
Среднее время загрузки

19 сек

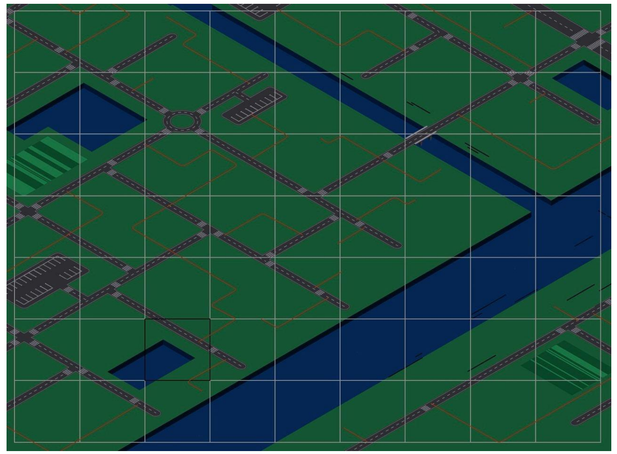
15 сек

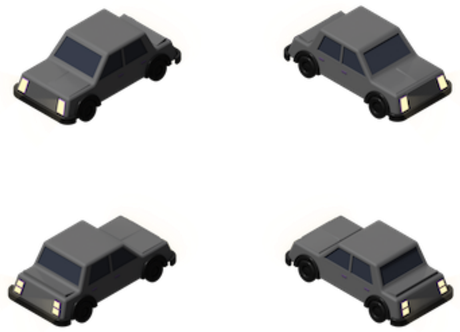
10 сек

Продумываем структуру игры



Что сделаем





Этап 1: Подключаем Phaser в React

- **Устанавливаем PhaserJS:**
  npm install phaser
- **Импортируем библиотеку :**
  import Phaser from 'phaser’
- **Создаем обёртку**
  игрового поля для React

```tsx
/** Обертка сцены с картой */
export const GameWrapper = () => {
    const [game, setGame] = useState<Phaser.Game>();
    const store = useStore();

    useEffect(() => {
        const scenes = [
            new BootScene(store),
            new MainScene(store)
        ];
        const config = getGameConfig(scenes);
        const _game = new Phaser.Game(config);

        setGame(_game);

        return (): void => {
            _game.destroy(true);
            setGame(undefined);
        };
    }, [store]);

    return <div id={YOOVILLAGE_CONTAINER_ID} />;
};

```

```
/** Возвращает конфиг сцены с картой */
export const getGameConfig = (scenes: Scene[]): Phaser.Types.Core.GameConfig => ({
    // автовыбор WebGl или Canvas
    type: Phaser.AUTO,
    // id контейнера где распологается карта
    parent: YOOVILLAGE_CONTAINER_ID,
    scene: scenes,
        parent: YOOVILLAGE_CONTAINER_ID,
        mode: Phaser.Scale.FIT,
        width,
        height
    },
    // убираем ограничение по текстурам
    maxTextures: -1,
    // здесь подключаем плагины, если необходимо
    plugins: {
        scene: [{
            key: 'rexGestures',
            plugin: GesturesPlugin,
            mapping: 'rexGestures'
        }]
    }
});
```

```
/** Возвращает конфиг сцены с картой */
export const getGameConfig = (scenes: Scene[]): Phaser.Types.Core.GameConfig => ({
    // автовыбор WebGl или Canvas
    type: Phaser.AUTO,
    // id контейнера где распологается карта
    parent: YOOVILLAGE_CONTAINER_ID,
    physics: {
        default: 'arcade',
        arcade: {
            gravity: {y: 0},
            // При разработке переключить в debug: true
            debug: false
        }
    },
        height
    },
    // убираем ограничение по текстурам
    maxTextures: -1,
    // здесь подключаем плагины, если необходимо
    plugins: {
        scene: [{
            key: 'rexGestures',
            plugin: GesturesPlugin,
            mapping: 'rexGestures'
        }]
    }
});
```

```
/** Возвращает конфиг сцены с картой */
export const getGameConfig = (scenes: Scene[]): Phaser.Types.Core.GameConfig => ({
    // автовыбор WebGl или Canvas
    type: Phaser.AUTO,
    // id контейнера где распологается карта
    parent: YOOVILLAGE_CONTAINER_ID,
    scene: scenes,
    physics: {
        default: 'arcade',
        arcade: {
            gravity: {y: 0},
            // При разработке переключить в debug: true
            debug: false
        }
    },
    scale: {
        parent: YOOVILLAGE_CONTAINER_ID,
        mode: Phaser.Scale.FIT,
        width,
        height
    },
        scene: [{
            key: 'rexGestures',
            plugin: GesturesPlugin,
            mapping: 'rexGestures'
        }]
    }
});
```

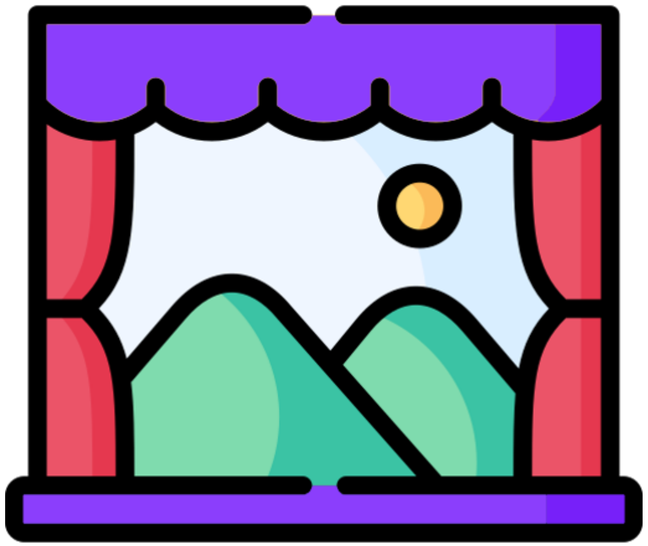
```
/** Возвращает конфиг сцены с картой */
export const getGameConfig = (scenes: Scene[]): Phaser.Types.Core.GameConfig => ({
    // автовыбор WebGl или Canvas
    type: Phaser.AUTO,
    // id контейнера где распологается карта
    parent: YOOVILLAGE_CONTAINER_ID,
    scene: scenes,
    physics: {
        default: 'arcade',
        arcade: {
            gravity: {y: 0},
            // При разработке переключить в debug: true
            debug: false
        }
    // убираем ограничение по текстурам
    maxTextures: -1,
    // здесь подключаем плагины, если необходимо
    plugins: {
        scene: [{
            key: 'rexGestures',
            plugin: GesturesPlugin,
            mapping: 'rexGestures'
        }]
    }
});
```

```ts
/** Возвращает конфиг сцены с картой */
export const getGameConfig = (scenes: Scene[]): Phaser.Types.Core.GameConfig => ({
    // автовыбор WebGl или Canvas
    type: Phaser.AUTO,
    // id контейнера где распологается карта
    parent: YOOVILLAGE_CONTAINER_ID,
    scene: scenes,
    physics: {
        default: 'arcade',
        arcade: {
            gravity: {y: 0},
            // При разработке переключить в debug: true
            debug: false
        }
    },
    scale: {
        parent: YOOVILLAGE_CONTAINER_ID,
        mode: Phaser.Scale.FIT,
        width,
        height
    },
    // убираем ограничение по текстурам
    maxTextures: -1,
    // здесь подключаем плагины, если необходимо
    plugins: {
        scene: [{
            key: 'rexGestures',
            plugin: GesturesPlugin,
            mapping: 'rexGestures'
        }]
    }
});
```

Этап 2: Создаём сцены

Методы сцены:

1. **preload** — загружаем тайловую карту, атласы, интерфейс загрузки
2. **create** — создаём все объекты, слои, группы, запускаем другие сцены
3. **update** — обновляем объекты




```ts
/** Сцена при которой загружаются файлы игры */
export class BootScene extends Phaser.Scene {
    store: Store<YooVillageState2, any>;
    mapTheme: ActualMapTheme;

    constructor(
        store: Store<YooVillageState2, any>,
    ) {
        super('BootScene');
        this.store = store;
    }
        this.loadTileMap();
        // Создаем прогресс-бар загрузки файлов игры
        this.createLoadingBar();
    }

    create() {
        // После загрузки запускаем главную сцену
        this.scene.start('MainScene');
    }

    update() {}
}
```

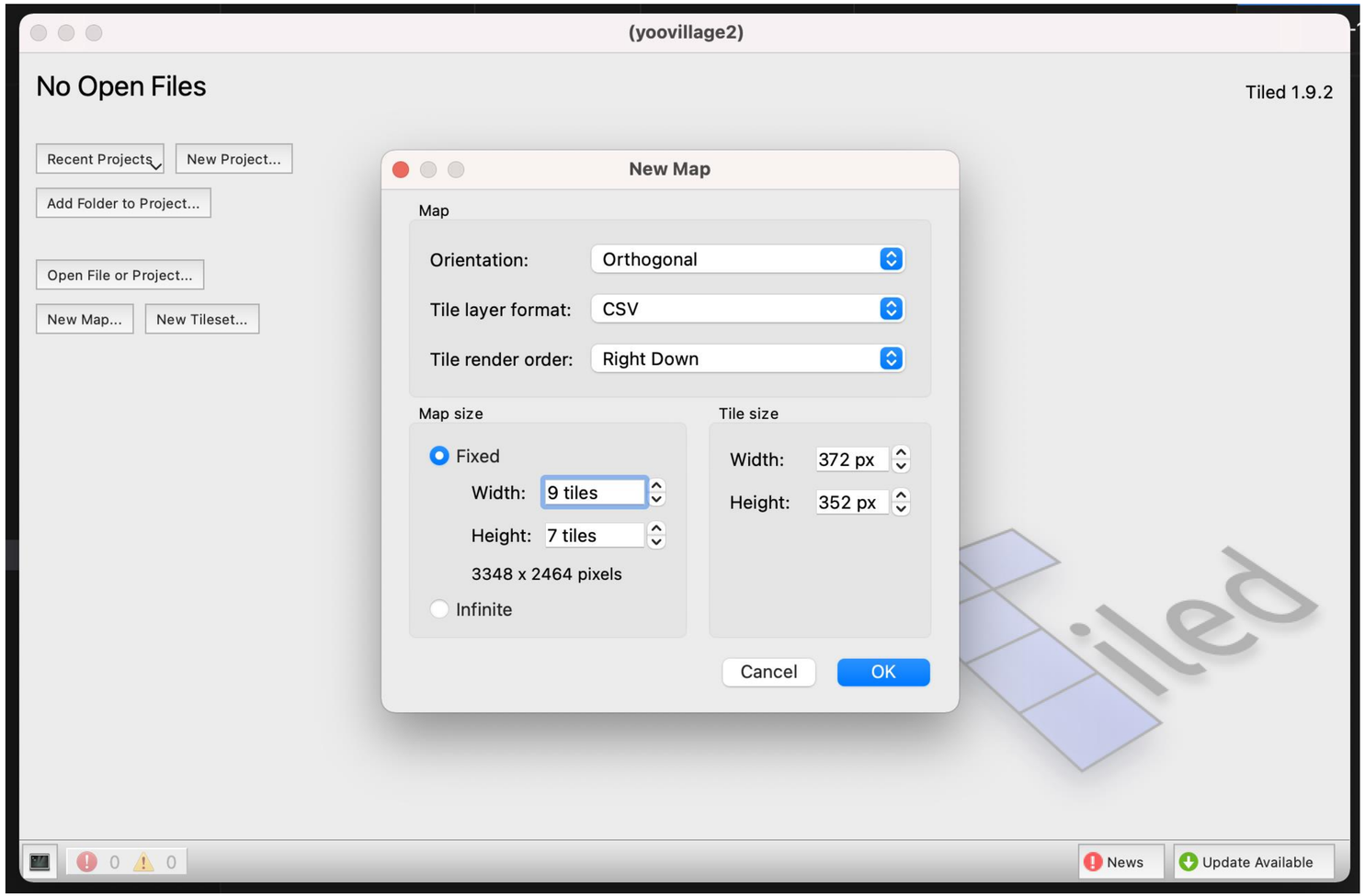

```
1  /** Сцена при которой загружаются файлы игры */
2  export class BootScene extends Phaser.Scene {
3      store: Store<YooVillageState2, any>;
4      mapTheme: ActualMapTheme;
5
12
13     preload() {
14         // Загружаем атлас с игровыми объектами
15         this.loadAtlas();
16         // Загружаем изображения
17         this.loadImages();
18         // Загружаем конфиги тайловых карт
19         this.loadTileMap();
20         // Создаем прогресс-бар загрузки файлов игры
21         this.createLoadingBar();
22     }
24     create() {
25         // После загрузки запускаем главную сцену
26         this.scene.start('MainScene');
27     }
28
29     update() {}
30 }
```


```
/** Сцена при которой загружаются файлы игры */
export class BootScene extends Phaser.Scene {
    store: Store<YooVillageState2, any>;
    mapTheme: ActualMapTheme;

    constructor(
        store: Store<YooVillageState2, any>,
    ) {
        super('BootScene');
        this.store = store;
    }

    preload() {
        // Загружаем атлас с игровыми объектами
        this.loadAtlas();
        // Загружаем изображения
        this.loadImages();
    create() {
        // После загрузки запускаем главную сцену
        this.scene.start('MainScene');
    }

    update() {}
}
```


```ts
/** Сцена при которой загружаются файлы игры */
export class BootScene extends Phaser.Scene {
    store: Store<YooVillageState2, any>;
    mapTheme: ActualMapTheme;

    constructor(
        store: Store<YooVillageState2, any>,
    ) {
        super('BootScene');
        this.store = store;
    }

    preload() {
        // Загружаем атлас с игровыми объектами
        this.loadAtlas();
        // Загружаем изображения
        this.loadImages();
        // Загружаем конфиги тайловых карт
        this.loadTileMap();
        // Создаем прогресс-бар загрузки файлов игры
        this.createLoadingBar();
    }

    create() {
        // После загрузки запускаем главную сцену
        this.scene.start('MainScene');
    }

    update() {}
}
```

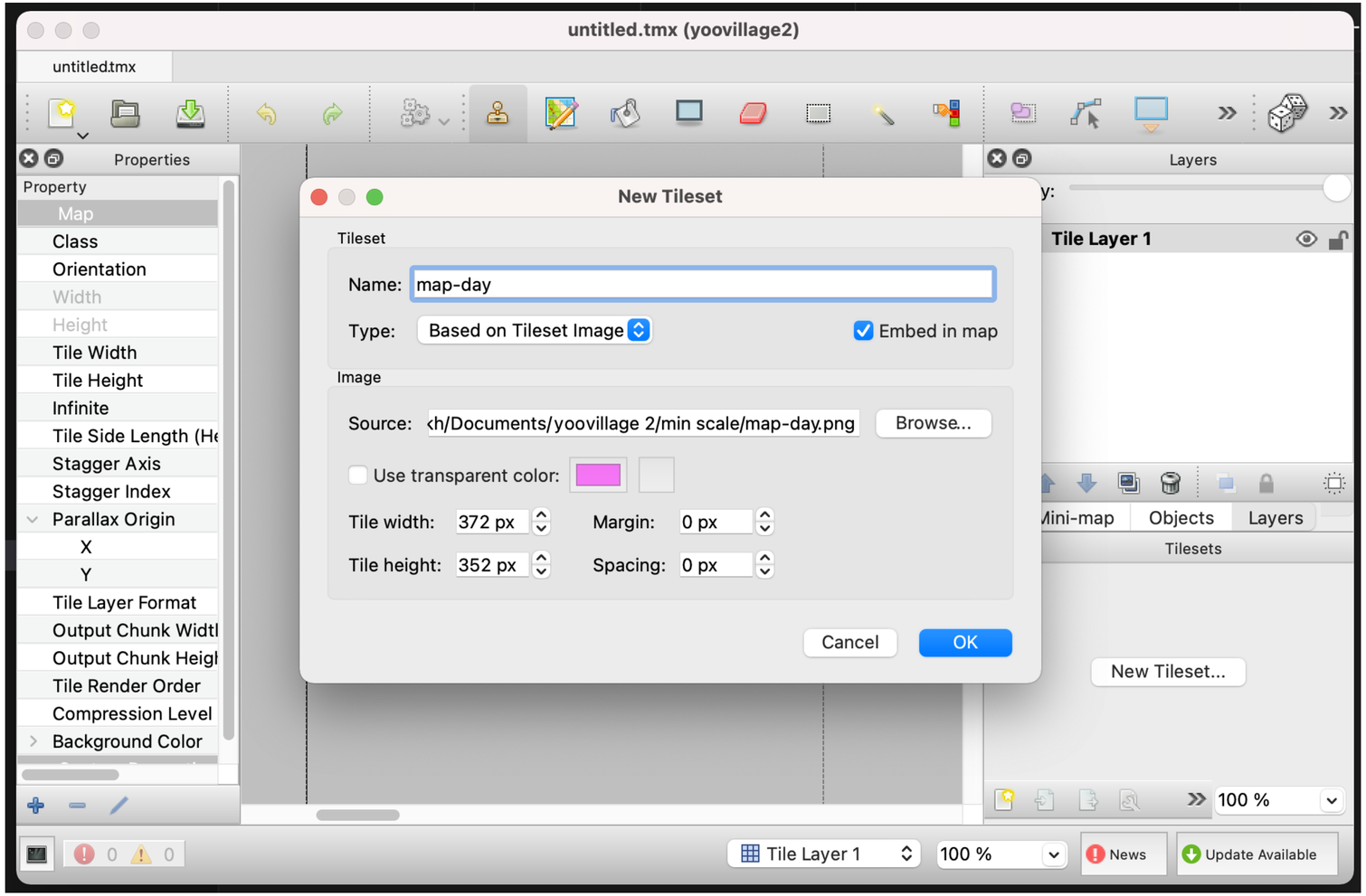
Этап 3: Создаём карту в Tiled

- Нажимаем “New Map”
- Выбираем количество тайлов по горизонтали и вертикали
- Выставляем ширину и высоту 1 тайла
- Нажимаем “ОК”

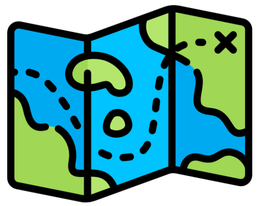


В открывшемся окне

- Нажимаем “New Tileset”
- Выбираем картинку карты
- Выставляем размер 1 тайла
- Нажимаем “ОК”

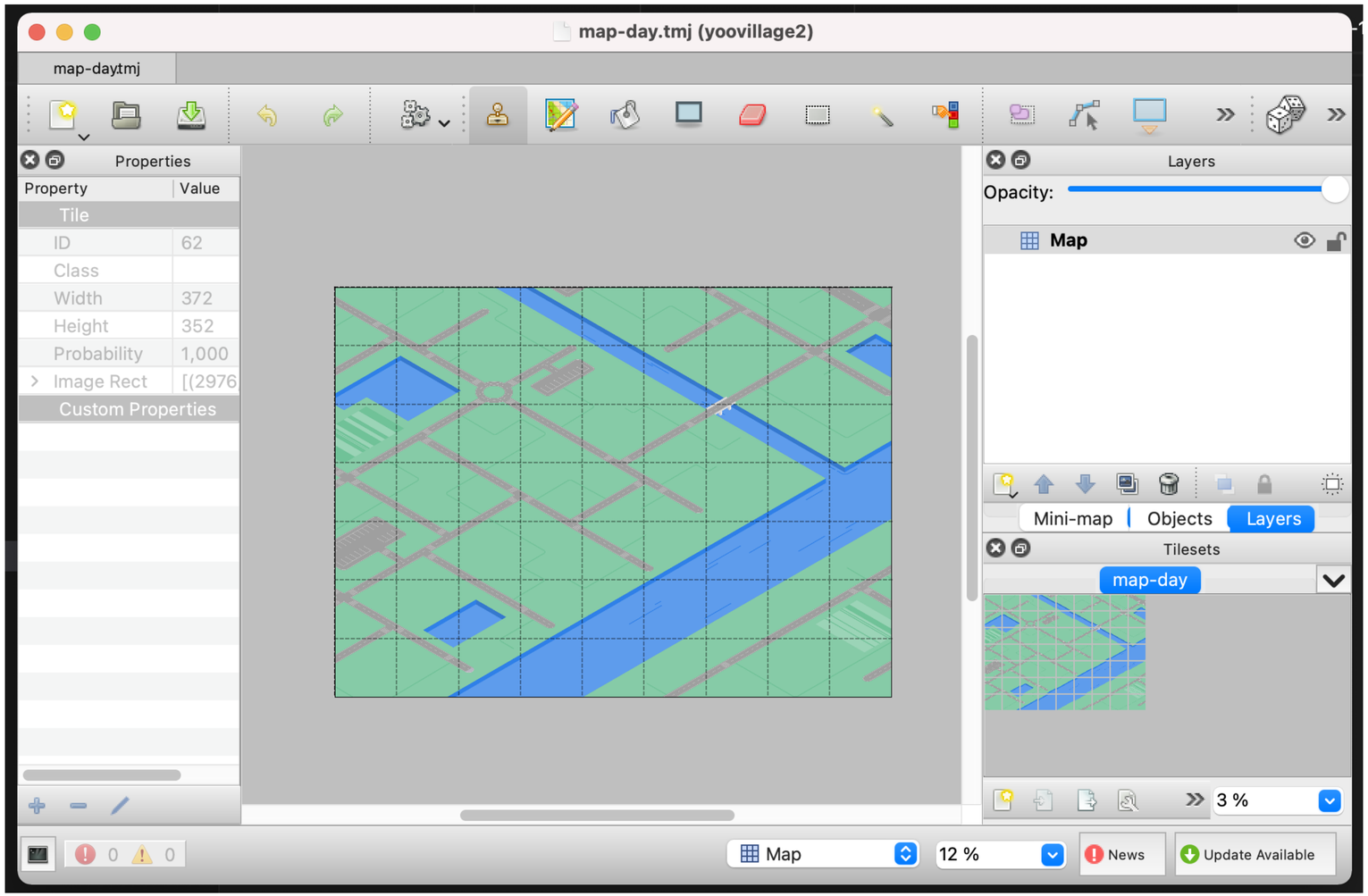


- Из квадратиков справа собираем полную карту
- сохраняем в формате .json

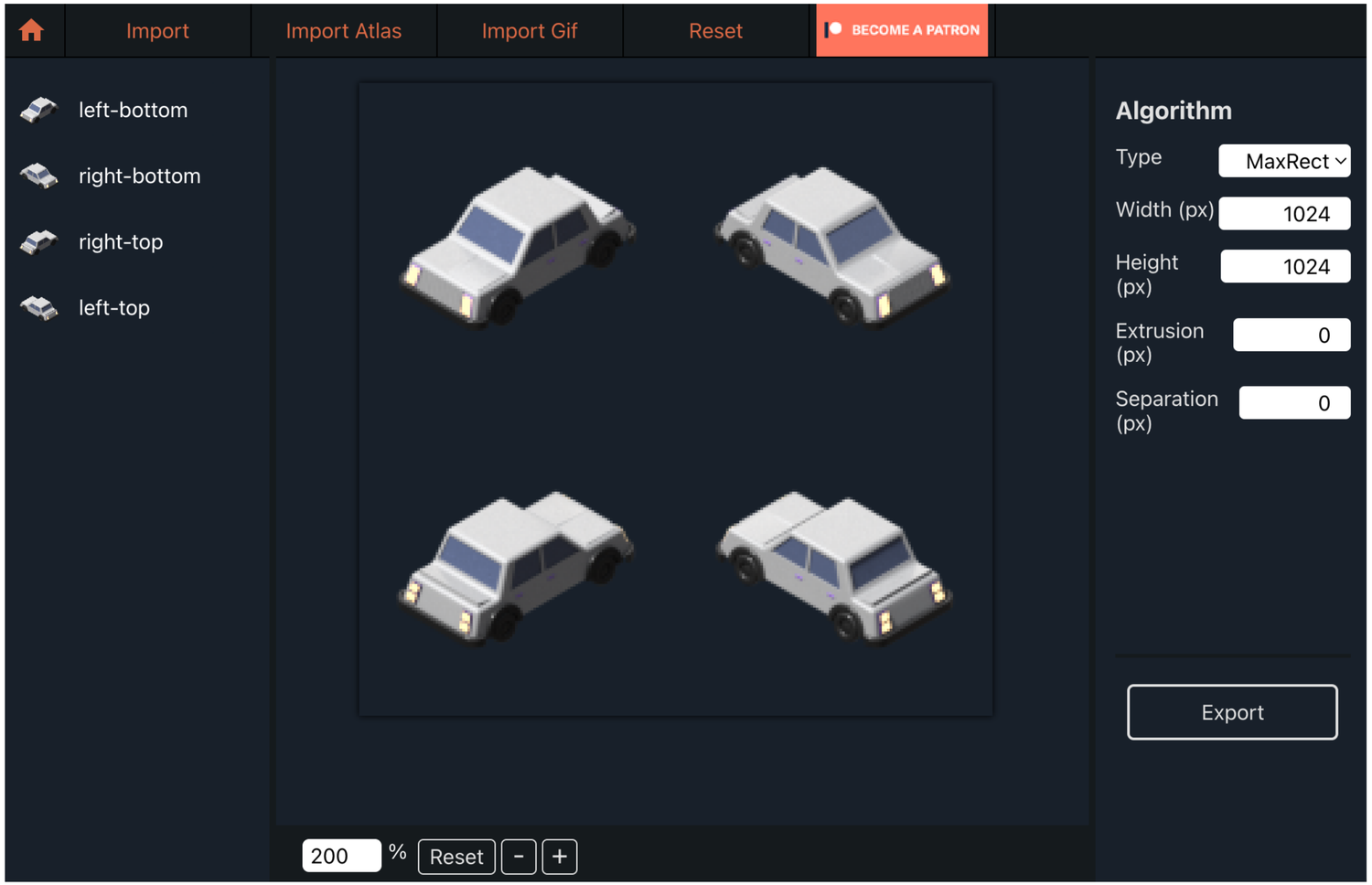


tiles-day.png

map-day.json



Этап 4: Создаём атласы в atlas-packer



car.png

car.json

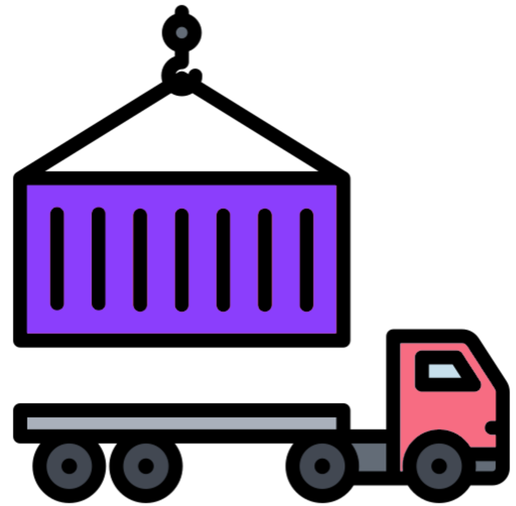
Этап 5: Загружаем файлы

Методы загрузки у сцены:

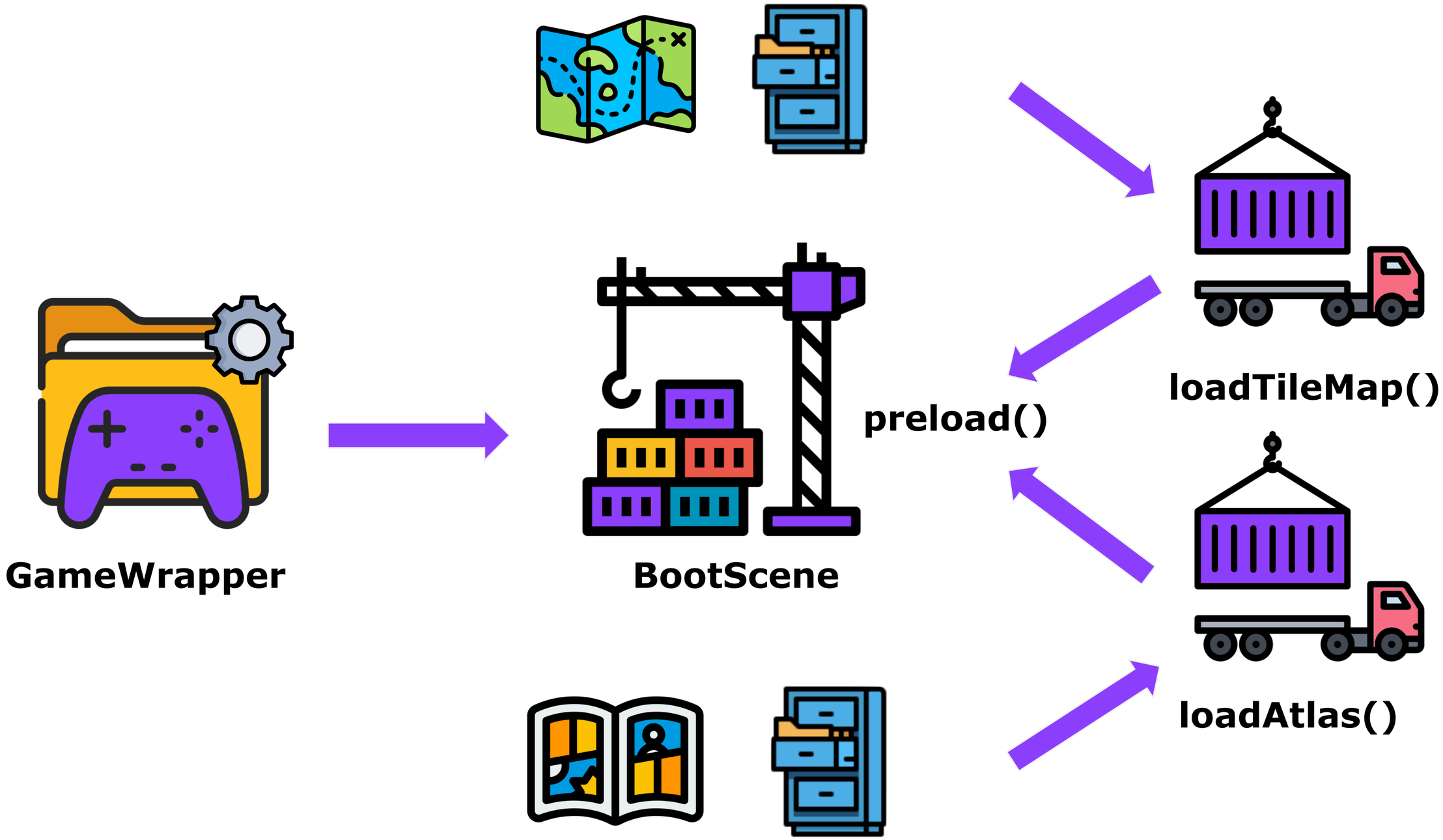
- **Загрузка изображений**
  `scene.load.image('image_name', image)`

- **Загрузка конфига к тайловой карте**
  `scene.load.tilemapTiledJSON('json_name', json)`

- **Загрузка атласа**
  `scene.load.atlas('atlas_name', atlasImage, atlasJson)`

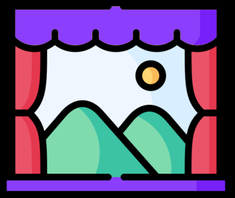


```
/** Загрузка тайловой карты */
loadTileMap() {
  // для дневной темы
    if (this.mapTheme === 'day') {
      this.load.image('tiles-day', tilesDayImage);
      this.load.tilemapTiledJSON('map-day', mapDayJson);
    }
  // для ночной темы
  if (this.mapTheme === 'night') {
      this.load.image('tiles-night', tilesNightImage);
      this.load.tilemapTiledJSON('map-night', mapNightJson);
    }
}
```


GameWrapper
BootScene
preload()
loadTileMap()
loadAtlas()

Этап 6: Создаём главную сцену с картой

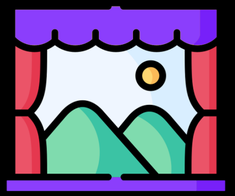
```
/** Главная сцена с картой */
export class MainScene extends Phaser.Scene {
    store: Store<YooVillageState, any>;
    state: YooVillageState;
    map: Tilemaps.Tilemap;

    constructor(store: Store<YooVillageState, any>) {
        super('MainScene');
        this.store = store;
        this.state = store.getState();
    }
      this.tiles = this.map.addTilesetImage(
        'map-key',
        `tiles-${this.state.mapTheme}`
      );
    }

    update() {}
}
```



Этап 6: Создаём главную сцену с картой

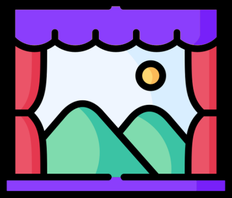
```
1 /** Главная сцена с картой */
2 export class MainScene extends Phaser.Scene {
3     store: Store<YooVillageState, any>;
4     state: YooVillageState;
```

```
13     create() {
14       // создаем карту
15       this.map = this.make.tilemap({key: 'map-key'});
16       // добавляем картинку тайлсета
17       this.tiles = this.map.addTilesetImage(
18         'map-key',
19         `tiles-${this.state.mapTheme}`
20       );
21     }
22
23     update() {}
24 }
```



Этап 6: Создаём главную сцену с картой

```
/** Главная сцена с картой */
export class MainScene extends Phaser.Scene {
    store: Store<YooVillageState, any>;
    state: YooVillageState;
    map: Tilemaps.Tilemap;

    constructor(store: Store<YooVillageState, any>) {
        super('MainScene');
        this.store = store;
        this.state = store.getState();
    }

    create() {
      // создаем карту
      this.map = this.make.tilemap({key: 'map-key'});
      // добавляем картинку тайлсета
      this.tiles = this.map.addTilesetImage(
        'map-key',
        `tiles-${this.state.mapTheme}`
      );
    }

    update() {}
}
```



Этап 7: Создаём классы для объектов

```
export class Car extends Phaser.Physics.Arcade.Sprite {
    scene: MainScene;
    id: string;
    x: number;
    y: number;
    group: Phaser.GameObjects.Group;

    constructor({scene, x, y, id, group}: CarProps) {
        super(scene, x, y, `car-atlas-name`, 'car-atlas-bottom-left-frame-name';
        this.scene = scene;
        this.x = x;
        this.y = y;
        this.id = id;
        this.group = group;

        // Добавляем машину в группу для машин в физический движок
        group.add(this);
    }
```

```
    update() {
        this.moveBottomRight();
    }
}
```

Этап 7: Создаём классы для объектов

```
export class Car extends Phaser.Physics.Arcade.Sprite {
    scene: MainScene;
    id: string;
    x: number;
    y: number;
    group: Phaser.GameObjects.Group;
```

```

    moveBottomRight() {
        this.setTexture(`car-atlas-name`, 'car-atlas-bottom-right-frame-name');
        this.x += 1;
        this.y += 1;
    }

    update() {
        this.moveBottomRight();
    }
}
```

Этап 7: Создаём классы для объектов

```
export class Car extends Phaser.Physics.Arcade.Sprite {
    scene: MainScene;
    id: string;
    x: number;
    y: number;
    group: Phaser.GameObjects.Group;

    constructor({scene, x, y, id, group}: CarProps) {
        super(scene, x, y, `car-atlas-name`, 'car-atlas-bottom-left-frame-name';
        this.scene = scene;
        this.x = x;
        this.y = y;
        this.id = id;
        this.group = group;

        // Добавляем машину в группу для машин в физический движок
        group.add(this);
    }

    moveBottomRight() {
        this.setTexture(`car-atlas-name`, 'car-atlas-bottom-right-frame-name');
        this.x += 1;
        this.y += 1;
    }

    update() {
        this.moveBottomRight();
    }
}
```

Этап 8: Добавляем все объекты на главную сцену

```
this.cars: Car[] = [];
// Добавляем группу для машин в физический движок
const carsGroup = this.physics.add.group();
const carsData = [
  {x: 0, y: 0},
  {x: 100, y: 200},
];

```

```
            })
            // сохраняем ссылку на машинку в сцене
            this.cars.push(newCar);
        })
    }

update() {
    // обновляем все машинки
    this.cars.forEach((car) => car.update());
}
```

Этап 8: Добавляем все объекты на главную сцену

```
 9 create() {
10         carsData.forEach((car, index) => {
11             const {x, y} = car;
12             const newCar = new Car({
13                 scene: this,
14                 x,
15                 y,
16                 id: `car-${index}`,
17                 group: carsGroup
18             })
19             // сохраняем ссылку на машинку в сцене
20             this.cars.push(newCar);
21         })
22     }
```

```
23
24 update() {
25     // обновляем все машинки
26     this.cars.forEach((car) => car.update());
27 }
```

Этап 8: Добавляем все объекты на главную сцену

```
this.cars: Car[] = [];
// Добавляем группу для машин в физический движок
const carsGroup = this.physics.add.group();
const carsData = [
  {x: 0, y: 0},
  {x: 100, y: 200},
];

create() {
        carsData.forEach((car, index) => {
            const {x, y} = car;
            const newCar = new Car({
                scene: this,
                x,
                y,
                id: `car-${index}`
```

```
update() {
    // обновляем все машинки
    this.cars.forEach((car) => car.update());
}
```

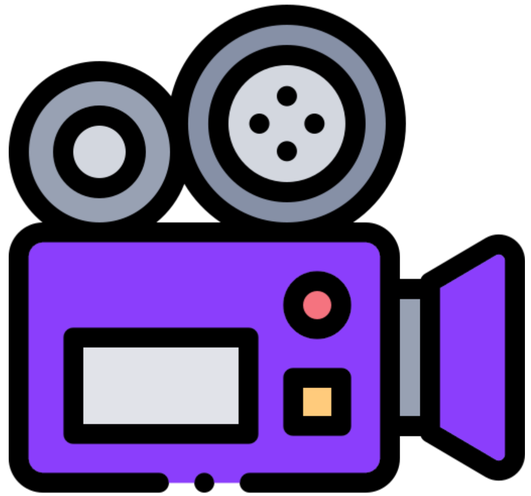
Этап 8: Добавляем все объекты на главную сцену

```
this.cars: Car[] = [];
// Добавляем группу для машин в физический движок
const carsGroup = this.physics.add.group();
const carsData = [
  {x: 0, y: 0},
  {x: 100, y: 200},
];

create() {
        carsData.forEach((car, index) => {
            const {x, y} = car;
            const newCar = new Car({
                scene: this,
                x,
                y,
                id: `car-${index}`,
                group: carsGroup
            })
            // сохраняем ссылку на машинку в сцене
            this.cars.push(newCar);
        })
    }

update() {
    // обновляем все машинки
    this.cars.forEach((car) => car.update());
}
```

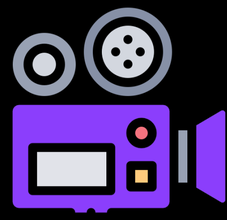
Этап 9: Настраиваем камеру

Создаем метод настройки камеры:

- Создание камеры
- Масштабирование
- Центрирование
- Настройка перемещения
- Настройка зума колесиком



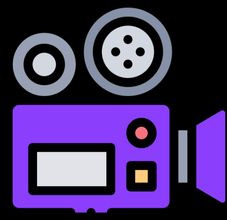
```
setupCamera() {
    const scene = this;
    // Настраиваем камеру
    const cam = scene.cameras.main;
    // Ограничиваем скролл по карте размерами карты
    cam.setBounds(
      0, 0,
      scene.map.map.widthInPixels,
      scene.map.map.heightInPixels
    );
    // Изначально центрируем камеру по центру карты
    cam.centerToBounds();

            if (newZoom > MIN_ZOOM) {
                cam.zoom = newZoom;
                scene.events.emit('zoom');
            }
        }
        if (deltaY < 0) {
            const newZoom = cam.zoom + ZOOM_STEP;
            if (newZoom < MAX_ZOOM) {
                cam.zoom = newZoom;
                scene.events.emit('zoom');
            }
        }
    });
```



```
setupCamera() {
    const scene = this;
    // Настраиваем камеру
    const cam = scene.cameras.main;
    // Ограничиваем скролл по карте размерами карты
    cam.setBounds(
      0, 0,
      scene.map.map.widthInPixels,
```

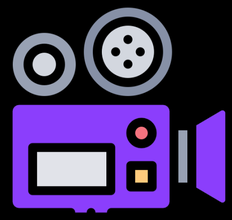
```
    // Настраиваем скролл при нажатии левой кнопки мыши и перетаскивании
    scene.input.on('pointermove', (p: Phaser.Input.Pointer) => {
        if (!p.isDown) {
            return;
        }

        cam.scrollX -= (p.position.x - p.prevPosition.x) / cam.zoom;
        cam.scrollY -= (p.position.y - p.prevPosition.y) / cam.zoom;
    });

```

```
            if (newZoom > MIN_ZOOM) {
                cam.zoom = newZoom;
                scene.events.emit('zoom');
            }
        }
        if (deltaY < 0) {
            const newZoom = cam.zoom + ZOOM_STEP;
            if (newZoom < MAX_ZOOM) {
                cam.zoom = newZoom;
                scene.events.emit('zoom');
            }
        }
    });
```

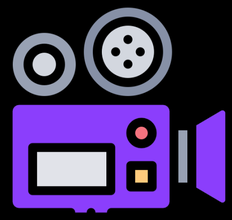


```
setupCamera() {
    const scene = this;
    // Настраиваем камеру
    const cam = scene.cameras.main;
    // Ограничиваем скролл по карте размерами карты
    cam.setBounds(
      0, 0,
      scene.map.map.widthInPixels,
      scene.map.map.heightInPixels
    );
```

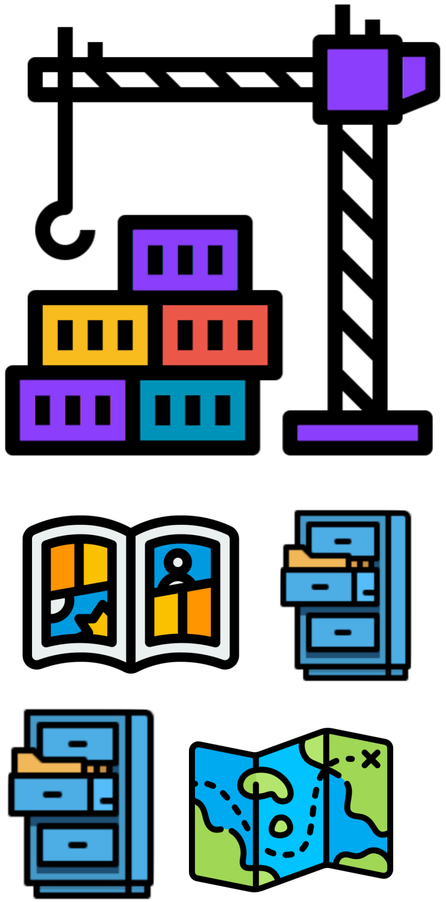
```
    // Настраиваем зумирование колесиком мыши
    scene.input.on('wheel', ({deltaY}: {deltaY: number}) => {
        if (deltaY > 0) {
            const newZoom = cam.zoom - ZOOM_STEP;
            if (newZoom > MIN_ZOOM) {
                cam.zoom = newZoom;
                scene.events.emit('zoom');
            }
        }
        if (deltaY < 0) {
            const newZoom = cam.zoom + ZOOM_STEP;
            if (newZoom < MAX_ZOOM) {
                cam.zoom = newZoom;
                scene.events.emit('zoom');
            }
        }
    });
```



```
setupCamera() {
    const scene = this;
    // Настраиваем камеру
    const cam = scene.cameras.main;
    // Ограничиваем скролл по карте размерами карты
    cam.setBounds(
      0, 0,
      scene.map.map.widthInPixels,
      scene.map.map.heightInPixels
    );
    // Настраиваем зум (по умолчанию 1)
    cam.setZoom(2);
    // Изначально центрируем камеру по центру карты
    cam.centerToBounds();

    // Настраиваем скролл при нажатии левой кнопки мыши и перетаскивании
    scene.input.on('pointermove', (p: Phaser.Input.Pointer) => {
        if (!p.isDown) {
            return;
        }

        cam.scrollX -= (p.position.x - p.prevPosition.x) / cam.zoom;
        cam.scrollY -= (p.position.y - p.prevPosition.y) / cam.zoom;
    });

    // Настраиваем зумирование колесиком мыши
    scene.input.on('wheel', ({deltaY}: {deltaY: number}) => {
        if (deltaY > 0) {
            const newZoom = cam.zoom - ZOOM_STEP;
            if (newZoom > MIN_ZOOM) {
                cam.zoom = newZoom;
                scene.events.emit('zoom');
            }
        }
        if (deltaY < 0) {
            const newZoom = cam.zoom + ZOOM_STEP;
            if (newZoom < MAX_ZOOM) {
                cam.zoom = newZoom;
                scene.events.emit('zoom');
            }
        }
    });
```

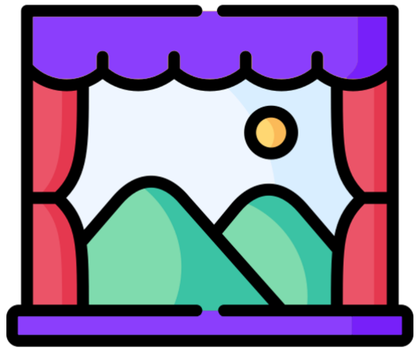


GameWrapper

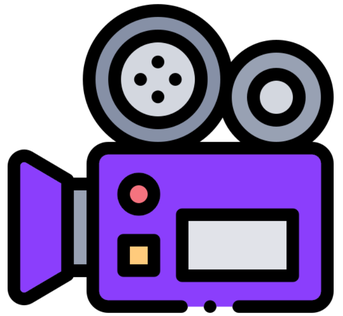


BootScene

class Car

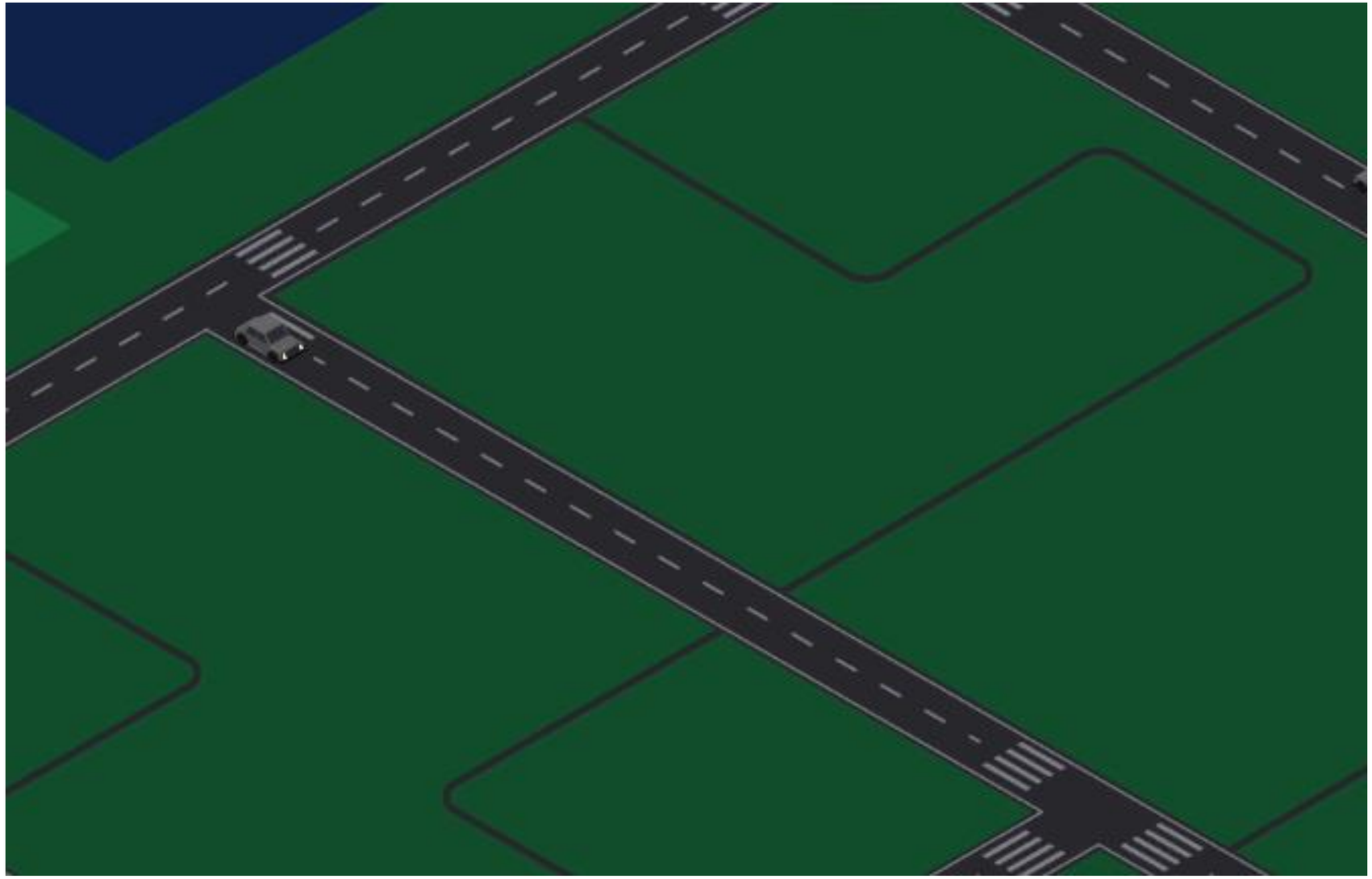


MainScene



setupCamera()

Что получилось



Сложности в разработке

Проблема 1

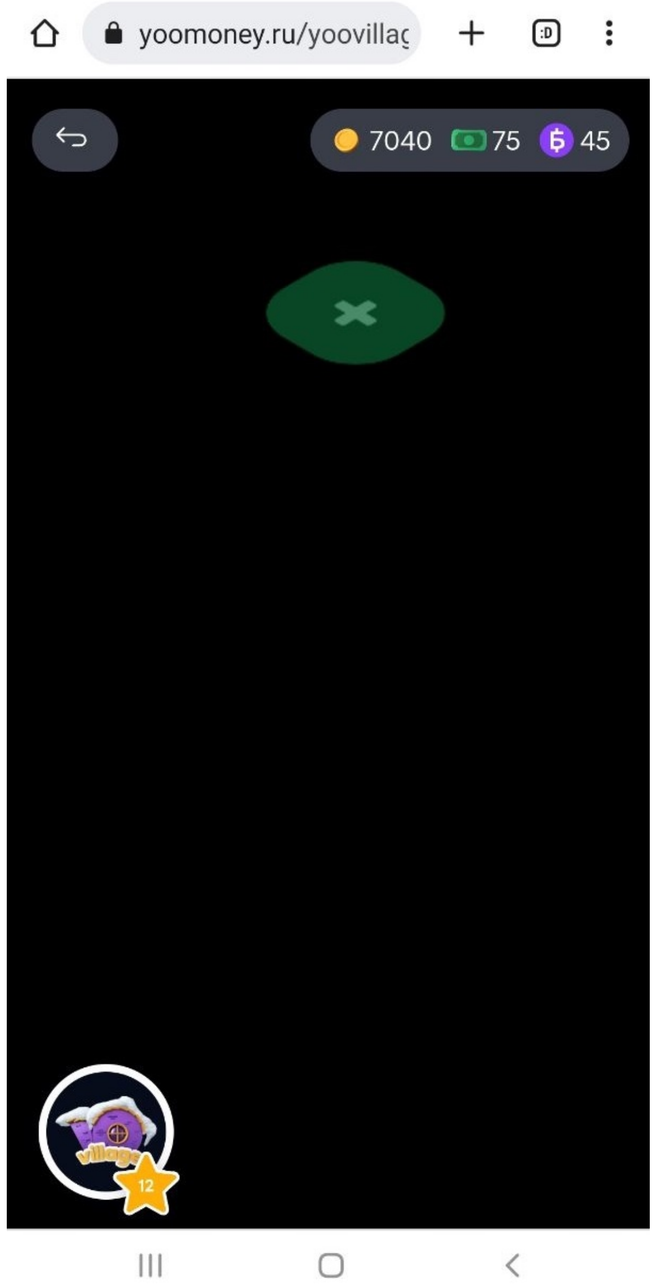
Игровая карта не отображалась
в Chrome на мобильных андроид-
устройствах

Причина

Ограничение на размер изображений.
Максимально допустимый размер картинки:
4096 * 4096 пикселей

Решение

Уменьшить разрешение карты



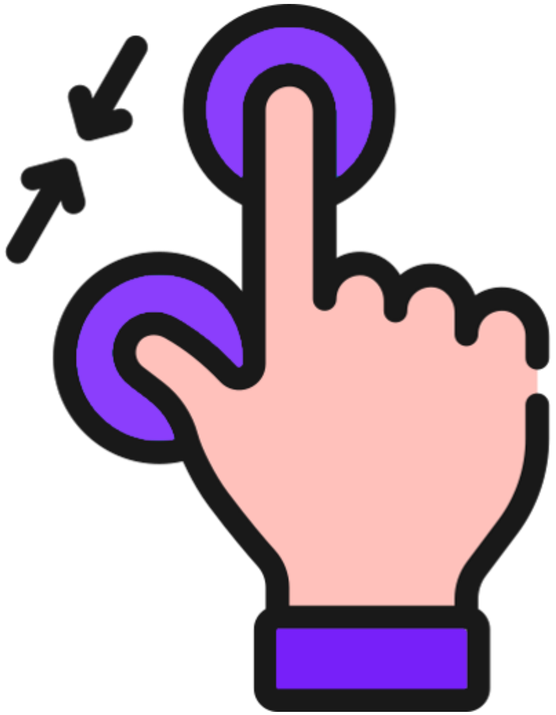
Проблема 2

В Phaser нет поддержки зума пинчем
на тачскрин-экранах

Решение

Использование плагина **phaser3-rex-plugins**, класс Pinch

```
// Настраиваем зумирование с помощью пинча 2 пальцами на мобильных устройствах
const pinch = new Pinch(this.scene);

pinch.on('pinch', function({scaleFactor}: Pinch) {
    const newZoom = cam.zoom * scaleFactor;

    if (newZoom > MIN_ZOOM && newZoom < MAX_ZOOM) {
        cam.zoom = newZoom;
        this.events.emit('zoom');
    }
}, this);
```

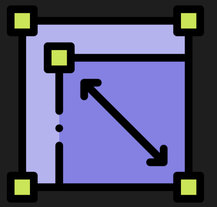


Проблема 3

Картинки на некоторых устройствах
сильно замылены

Причина:
Высокий Device Pixel Ratio устройства
(Phaser не поддерживает DPR)

Решение:
При зуме и изменении размера экрана
вручную пересчитываем ширину и высоту с
учётом DPR



```ts
export const rescaleHandler = (scene: MainScene | InitScene) => {
    const widthDPR = Math.round(window.innerWidth * DPR);
    const heightDPR = Math.round(window.innerHeight * DPR);

    if (scene.scale.parent) {
        scene.scale.parent.width = Math.round(window.innerWidth);
        scene.scale.parent.height = Math.round(window.innerHeight);
    }

    if (scene.scale.canvas) {
        scene.scale.canvas.width = widthDPR;
        scene.scale.canvas.height = heightDPR;

        scene.scale.canvas.style.width = `${Math.round(window.innerWidth)} + px`;
        scene.scale.canvas.style.height = `${Math.round(window.innerHeight)} + px`;
    }

    scene.scale.setGameSize(widthDPR, heightDPR);
    scene.scale.setParentSize(window.innerWidth, window.innerHeight);
};

```

Проблема 4

Текст внутри Phaser на некоторых устройствах сильно замылен

Причина
По умолчанию текст установлен с низким разрешением для повышения производительности

Решение
Увеличить разрешение текста вручную, вызвав встроенный метод
setResolution(2)

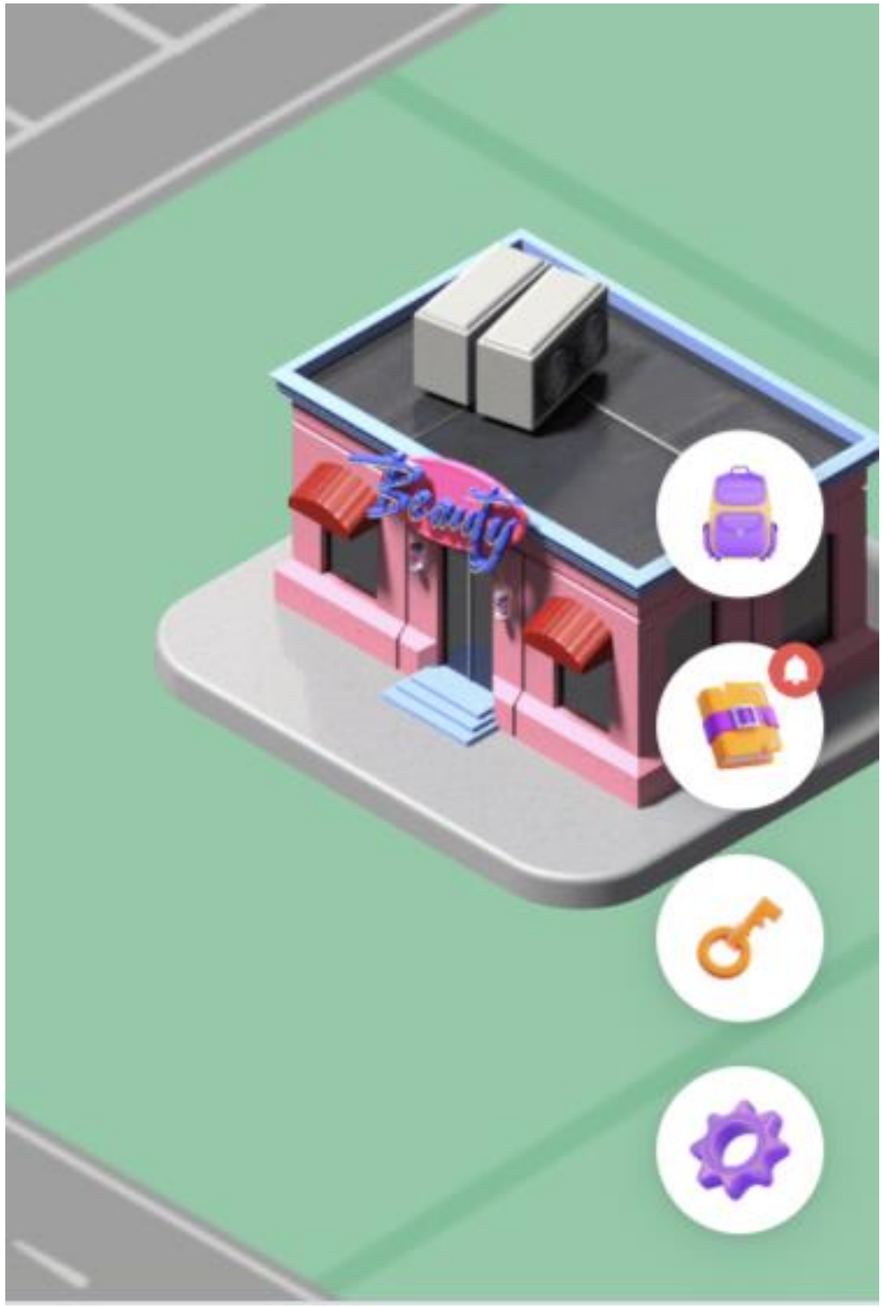
```
structure.popoverText = scene.make.text({
    x,
    y,
    text: 'Тут текст',
    style: {
      fontFamily: 'Factor IO',
      fontSize: '16px',
      color: '#000',
      padding: {x: 12, y: 8}
    }
})
    // центрируем текст горизонтально
    .setOrigin(0.5, 0)
    // увеличиваем четкость текста чтобы он не размывался
    .setResolution(2);
```


Тыква


Тыква

Проблема 5

Сквозные клики через меню



Решение

Добавление проверки, был ли клик
непосредственно по канвасу игровой карты

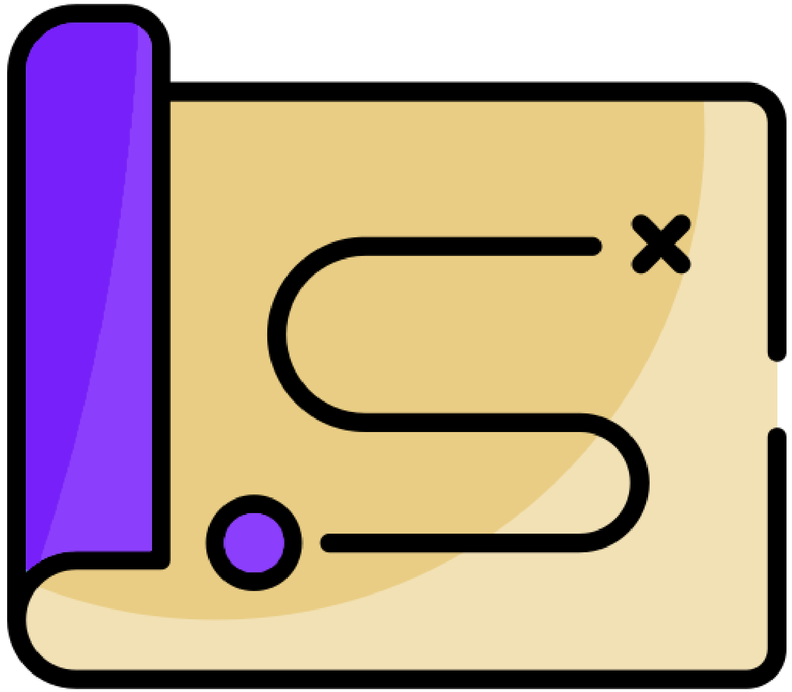
```
1     car.on('pointerdown', (pointer: Phaser.Input.Pointer) => {
2         // Проверяем что клик идет по канвасу
3         // и в случае если интерфейс накрывает его - ничего не делаем
4         if (pointer.downElement === document.getElementsByTagName('canvas')[0]) {
5             // создает событие которое обрабатывается в GameManager
6             this.scene.events.emit('clickOnCar');
7         }
8     });
```

Проблема 6

При перемещении по карте срабатывают лишние клики по объектам

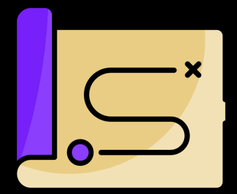
Решение

Добавление проверки на «протаскивание»



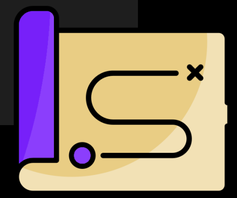
```
export const checkIsEventDragging = (
  {oldX, oldY, x, y, event,}: IsEventDraggingProps
) => {
    if (Math.abs(oldX - x) + Math.abs(oldY - y) < 10) {
        return event();
    }
};
```

```
    sprite.on('pointerup', (pointer: Phaser.Input.Pointer) => {
        const {editModeEnabled} = this.scene.state.village;
        checkIsEventDragging({
            oldX: this.pointerX,
            oldY: this.pointerY,
            x: pointer.position.x,
            y: pointer.position.y,
            event: () => this.clickHandler(pointer)
        });
    });
}
```

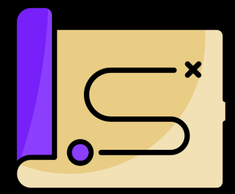


```
export const checkIsEventDragging = (
  {oldX, oldY, x, y, event,}: IsEventDraggingProps
) => {
```

```
addClickHandler(sprite: Phaser.Physics.Arcade.Sprite | Phaser.GameObjects.Sprite) {
    sprite.on('pointerdown', (pointer: Phaser.Input.Pointer) => {
        this.pointerX = pointer.position.x;
        this.pointerY = pointer.position.y;
    });

    sprite.on('pointerup', (pointer: Phaser.Input.Pointer) => {
        const {editModeEnabled} = this.scene.state.village;
        checkIsEventDragging({
            oldX: this.pointerX,
            oldY: this.pointerY,
            x: pointer.position.x,
            y: pointer.position.y,
            event: () => this.clickHandler(pointer)
        });
    });
}
```



```
export const checkIsEventDragging = (
  {oldX, oldY, x, y, event,}: IsEventDraggingProps
) => {
    if (Math.abs(oldX - x) + Math.abs(oldY - y) < 10) {
        return event();
    }
};

addClickHandler(sprite: Phaser.Physics.Arcade.Sprite | Phaser.GameObjects.Sprite) {
    sprite.on('pointerdown', (pointer: Phaser.Input.Pointer) => {
        this.pointerX = pointer.position.x;
        this.pointerY = pointer.position.y;
    });

    sprite.on('pointerup', (pointer: Phaser.Input.Pointer) => {
        const {editModeEnabled} = this.scene.state.village;
        checkIsEventDragging({
            oldX: this.pointerX,
            oldY: this.pointerY,
            x: pointer.position.x,
            y: pointer.position.y,
            event: () => this.clickHandler(pointer)
        });
    });
}
```



Что получили

- Возможность спрайтовой анимации
- Удобная работа с объектами
- Встроенный физический движок
- Улучшение графики
- Обширный инструментарий
- Удобная работа со слоями
- Быстрый плавный скролл

Вот что у нас получилось

Спасибо
за внимание :)