📖 ПРАКТИЧЕСКОЕ РУКОВОДСТВО: КАК СОБИРАТЬ ГРАФЫ AI-АГЕНТОВ

В этой инструкции мы пройдем от самых простых до сложных сценариев.

Золотое правило редактора:

Чтобы система поняла, что связь (стрелочка) между узлами управляет потоком выполнения, на всех таких связях обязательно должен стоять тег _aCall.

Вариант 1. Одиночный Агент (Проверка связи)

Самый простой сценарий. Пользователь задает вопрос, агент отвечает.

1. Создайте Главный узел (Root)
- _agentType = Sequential
- Это стартовая точка. При запуске графа вы указываете ID именно этого узла.
2. Создайте Узел-Агент
- _agentType = Agent
- _agent = Пират (имя)
- _guidance = Ты пират. Отвечай на все вопросы с пиратским акцентом.
3. Связи (Edges)
- Проведите связь от Главного узла к Агенту.
- Добавьте тег связи: _aCall.

Вариант 2. Линейный конвейер (Писатель -> Редактор)

Ответ первого агента автоматически передается второму. Без сложных сохранений.

1. Главный узел (Root)
- _agentType = Sequential
2. Узел 1 (Писатель)
- _agentType = Agent
- _guidance = Напиши короткий рассказ на тему, которую задаст пользователь.
3. Узел 2 (Редактор)
- _agentType = Agent
- _guidance = Ты строгий редактор. Исправь ошибки.
- _inputTemplate = Вот текст от автора: {Input}. Исправь его.
4. Связи (Edges)
- От Root к Писателю (Тег: _aCall, Свойство Weight = 10).
- От Root к Редактору (Тег: _aCall, Свойство Weight = 5).
- Примечание: Sequential узел запускает дочерние узлы по очереди, от большего Weight к меньшему.

Вариант 3. Умное ветвление (Маршрутизатор)

Если вопрос про код — отвечает Программист, если про деньги — Бухгалтер.

1. Главный узел (Root)
- _agentType = Sequential
2. Узел 1 (Классификатор)
- _agentType = Agent
- _guidance = Ответь ровно одним словом: "tech", если вопрос про IT, или "biz", если про бизнес.
- _saveStateKey = category (сохраняет метку в память)
- _forwardInput = true (передает оригинальный вопрос дальше, а не слово tech/biz)
3. Узел 2 (Маршрутизатор)
- _agentType = Conditional
- _routeKey = category (смотрит в память)
4. Узлы 3 и 4 (Эксперты)
- Создайте два обычных узла Agent (один Программист, другой Бухгалтер).
5. Связи (Edges)
- От Root к Классификатору (_aCall, Weight = 10).
- От Root к Маршрутизатору (_aCall, Weight = 5).
- От Маршрутизатора к Программисту (_aCall, Свойство связи _routeValue = tech).
- От Маршрутизатора к Бухгалтеру (_aCall, Свойство связи _routeValue = biz).

Вариант 4. Параллельная работа (Одновременная проверка)

Один текст отправляется двум критикам одновременно. Их ответы склеиваются в один текст.

1. Главный узел (Root)
- _agentType = Sequential
2. Узел 1 (Автор)
- _agentType = Agent
- _guidance = Напиши 3 предложения на заданную тему.
- _saveStateKey = draft (сохраняем черновик)
3. Узел 2 (Менеджер параллельности)
- _agentType = Parallel
4. Узлы 3 и 4 (Критик фактов и Критик стиля)
- Оба узла: _agentType = Agent.
- Оба узла: _inputTemplate = Оцени черновик: {State:draft}.
5. Связи (Edges)
- От Root к Автору (_aCall, Weight=10).
- От Root к Менеджеру параллельности (_aCall, Weight=5).
- От Менеджера параллельности к Критику 1 и Критику 2 (на обеих тег _aCall).

Вариант 5. Использование Инструментов (Web Search)

Агент должен сходить в интернет перед тем, как ответить.

1. Главный узел (Root) -> _agentType = Sequential
2. Узел (Исследователь)

- _agentType = Agent
- _guidance = Ответь на вопрос пользователя. Всегда ищи актуальную информацию в сети.
- _planning = ReAct (Критически важно! Это дает агенту "мозги" для использования функций)
- websearch = true (Включает инструмент)
- _maxTokens = 2000 (ReAct требует много токенов на рассуждения)

3. Связи (Edges)

- От Root к Исследователю (_aCall).

Вариант 6. Шедевр (Сложный вложенный граф)

Объединяем всё: Автор пишет черновик -> Двое критикуют параллельно -> Финальный редактор собирает всё вместе.

1. Главный узел (Root) Sequential
2. Узел 1 (Писатель)

- _agentType = Agent, _saveStateKey = expert_draft.

3. Узел 2 (Менеджер Параллельности)

- _agentType = Parallel.

4. Узлы 3 и 4 (Ревьюеры)

- Подключаются связями к Узлу 2 (Менеджеру Параллельности).
- _inputTemplate = Проверь черновик: {State:expert_draft}.

5. Узел 5 (Финальный Редактор)

- _agentType = Agent.
- _inputTemplate = Оригинальный вопрос: {Original}. Черновик: {State:expert_draft}. Отзывы критиков: {Input}. Напиши идеальный финальный вариант.
- (Секрет: В {Input} здесь автоматически попадут склеенные ответы от Ревьюеров, так как Редактор стоит после Parallel-узла).

6. Связи от Root (по порядку выполнения):

- Root -> Писатель (Weight = 30)
- Root -> Менеджер Параллельности (Weight = 20)
- Root -> Финальный Редактор (Weight = 10)

📌 Краткая шпаргалка для тестировщиков

- Как передать текст дальше? Используйте Sequential и не пишите ничего в _saveStateKey. Текст передастся автоматически как {Input}.

- Как передать текст "через голову" (например, от 1-го узла к 5-му)? В 1-м узле напишите _saveStateKey = my_text. В 5-м узле напишите _inputTemplate = {State:my_text}.
- Узел сработал, но съел вопрос? Если агент классифицирует текст (выдает "tech"), то следующий агент получит слово "tech". Чтобы следующий получил изначальный вопрос, поставьте в классификаторе _forwardInput = true.
- Ошибка нехватки контекста? Поставьте на стартовом (Главном) узле _maxTokens = 2048.
- Ничего не происходит? Проверьте, есть ли тег _aCall на стрелочках между узлами. Сборщик игнорирует стрелки без этого тега!

Руководство по системе управления, автоматизации и телеметрии 3D-роботов в Socrates Studio

Вся робототехника в Socrates Studio построена на единой шине данных: «Нейросеть (ИИ) ⇄ Скретч (Blockly) ⇄ 3D-Полигон (Babylon.js + Havok)».

1. Сводная таблица каналов управления

Способ управления	Источник	Назначение	Формат команды / Инструмент
Скретч-логика (Blockly)	Клиентский UI / Скрипт	Локальная автоматизация, реакция на датчики, последовательности	Визуальные кубики категории 🤖 3D Робототехника
Команды от ИИ (LLM)	Socrates AI / Нейросеть	Высокоуровневые решения, автономное поведение	JSON / Текстовые директивы через WebSocket
SHP-скрипты в гБД	Сервер ГНС / Узлы графа	Программная логика в нейронах RealObjectNeuron	Let("(Robot:Arm {_action='animateJoint', ...})")
Прямой JS API	JavaScript / WebView	Низкоуровневый доступ к физике Havok	Объект window.Socrates3D.*

2. Вариант А: Управление через Скретч (Blockly-кубики)

Кубики находятся во вкладке 🧩 Логика в категории 🤖 3D Робототехника:

1. Переключение режимов экрана

- Кубик: [🖥️ Переключить экран на: [🤖 3D Полигон ▼ / 🎨 Клиентский UI ▼]]
- Назначение: скрывает 2D-интерфейс и разворачивает полигон на весь экран (или наоборот).

2. Плавная кинематическая анимация сустава

- Кубик: [🎬 Плавно повернуть узел ID [servo_1] до угла [90°] за [1.5] сек]

- Назначение: плавно интерполирует угол поворота рычага/манипулятора без рывков с сохранением физики Havok.

3. Автоматическое движение шасси

- Кубик: [🚜 Движение робота: [Вперед ▼ / Назад ▼ / Поворот влево ▼ / Поворот вправо ▼ / Стоп ▼] мощность: [60%] время: [2.0] сек]
- Назначение: раскручивает все колёса/гусеницы робота на заданное время, затем автоматически останавливает.

4. Автономный триггер дальномера (Событие)

- Кубик: [📡 Когда дистанция на дальномере ID [sensor_1] < [20] см -> Выполнить [...]]
- Назначение: запускает вложенные блоки, как только перед роботом появляется препятствие ближе указанного расстояния.

5. Клешня-захват

- Кубик: [💪 Клешня захвата ID [gripper_1] раскрыть (%): [50]]
- Назначение: открывает/сжимает захват от 0% (полностью сжат) до 100% (максимально раскрыт).

6. Снимок с бортовой камеры и отправка в ИИ

- Кубик: [📷 Сделать фото с камеры и отправить в ИИ]
- Назначение: захватывает текущий кадр с камеры робота и шлёт в мультимодальный канал ИИ для зрительного анализа.

3. Вариант Б: Текстовые команды от ИИ (JSON-протокол)

ИИ-модель Socrates может отправлять управляющие команды прямо в тексте своих ответов. Диспетчер window.Socrates3D.executeRobotCommand(cmd) парсит их на лету.

Примеры JSON-директив от ИИ:

1. Движение робота

```
codeJSON
{ "action": "drive", "dir": "FORWARD", "power": 80, "dur": 3.0 }
```

- dir: "FORWARD" (вперед), "BACKWARD" (назад), "LEFT" (влево), "RIGHT" (вправо), "STOP" (стоп).
- power: мощность от 0 до 100.
- dur: длительность в секундах.

2. Плавный поворот манипулятора / сервомотора

```
codeJSON
{ "action": "animateJoint", "id": "servo_arm_1", "angle": 45, "duration": 1.2 }
```

3. Управление захватом (Клешней)

```
codeJSON
{ "action": "setGripper", "id": "gripper_main", "val": 0 }
```

(0 = захватить банку, 100 = отпустить).

4. Запрос фотоснимка окружающей обстановки

```
codeJSON
{ "action": "captureSnapshot" }
```

4. Вариант В: Автономная телеметрия (Робот ➔ ИИ)

Робот может самостоятельно информировать нейросеть о своём состоянии без внешнего запроса:

1. Дистанция с сонара/лидара:

```
codeJavaScript
window.Socrates3D.sendTelemetry("ultrasonic_sensor_1", "Впереди стена: 14.5 см. Куда ехать?");
```

2. Снимок камеры по событию:

```
codeJavaScript
window.Socrates3D.sendCameraSnapshot();
```

3. Ошибки и перевороты:

Если угол наклона базы робота превышает 60°, робот шлёт сигнал бедствия в канал ошибок _Error.

5. Вариант Г: Прямой низкоуровневый JavaScript API (window.Socrates3D)

Для использования в кастомных скриптах или веб-виджетах:

```
codeJavaScript
// 1. Мгновенно задать угол сервомотора (в градусах)
window.Socrates3D.setJointAngle("servo_1", 90);

// 2. Плавно повернуть за 2 секунды
window.Socrates3D.animateJoint("servo_1", 90, 2.0);

// 3. Задать угловую скорость отдельного колеса (рад/с)
window.Socrates3D.setWheelSpeed("wheel_left", 12.0);

// 4. Считать дистанцию до ближайшего объекта (в см)
var dist = window.Socrates3D.getDistance("ultrasonic_1");

// 5. Включить / выключить лазерный луч дальномера
window.toggleSensorLaser(mesh);

// 6. Переключить камеру в режим «глаза робота» (FPV)
```

```
window.toggleFpvCamera();

// 7. Старт / пауза физической симуляции Havok (Play / CAD)
window.toggleSimulationMode();
```

6. Типовой сценарий «Умный робот-доставщик»

1. Пользователь (в чате): «Привези банку со стола».
2. ИИ (LLM) переключает вид на полигон:

```
window.SocratesBridge.switchViewMode('polygon')
```

3. ИИ запускает шасси вперед:

```
{"action": "drive", "dir": "FORWARD", "power": 70, "dur": 2.5}
```

4. Сенсор робота видит банку на дистанции < 15 см ➔ срабатывает триггер robot_on_obstacle_near ➔ робот тормозит и делает снимок:

```
window.Socrates3D.sendCameraSnapshot()
```

5. ИИ анализирует снимок, подтверждает цель и закрывает клешню:

```
{"action": "setGripper", "id": "gripper_1", "val": 0}
```

6. Робот разворачивается и возвращается назад.

```
14.9s
info
Google AI models may make mistakes, so double-check outputs.
Use Arrow Up and Arrow Down to select a turn, Enter to jump to it, and Escape to return to the chat.
```