2.2.3. Отладка игры. Системный поиск багов и оптимизация логики

Блок: Блок 2.2. Программирование (Продвинутый) · **Полугодие:** 2 · **Класс:** 5-7 · **Продолжительность:** 40 мин

Тема: Дебаггинг: консоль, классификация багов, оптимизация · **Инструменты:** ПРИМС (Cyberiada Graph Editor), Мастерская игр «Урсула», Inky

---

📋 Паспорт урока

🎯 Цель	Сформировать у учащихся культуру системного поиска и исправления ошибок (дебаггинга) в игровых проектах. Научить читать системные сообщения в консоли браузера, применять индикацию состояний через МодульИндикации, использованию классификации багов по критичности и приёмам оптимизации логики поведения.
📚 Тип урока	Урок-практикум по системной отладке игровых проектов.
👥 Формы работы	Фронтальная демонстрация методов поиска ошибок, индивидуальная практическая работа над устранением багов в своих проектах и баг-листом.
🛠️ Оборудование	Проекты учеников, редакторы ПРИМС/Inky, «Урсула», проектор.

🎯 Планируемые результаты

Предметные

- Учащиеся знают классификацию багов: критичные, важные, мелкие.

- Учащиеся умеют открывать и читать консоль разработчика в браузере для отслеживания событий движка и смены состояний.
- Учащиеся владеют методом визуальной отладки через цветовую маркировку нод (МодульИндикации).
- Учащиеся знают приёмы оптимизации: упрощение диаграмм, устранение циклов.
- Учащиеся применяют правило «одной правки за раз» при тестировании гипотез.

Метапредметные

Коммуникативные	• Умение точно и воспроизводимо описывать обнаруженную ошибку без эмоциональных оценок.
Регулятивные	• Работа по баг-листу с приоритетами. • Самоконтроль при внесении изменений: отказ от хаотичного редактирования рабочей логики в пользу последовательной проверки гипотез.
Познавательные	• Анализ причинно-следственных связей в логике. • Применение гипотетико-дедуктивного метода: сопоставление видимого поведения объекта на сцене с внутренним состоянием автомата в графе.

📚 Ход урока

Этап 1. Введение (2 мин)

🖥️ Слайд 1: Титульный

Подзаголовок: «Баги есть у всех. Уметь их искать —важный навык разработчика».

📣 Действия учителя: Приветствие.

Тема: сегодня учимся искать и чинить баги системно.

«В сложных играх, где десятки объектов обмениваются сигналами, что-то неизбежно идет не так: робот замер на месте, терминал перестал отвечать, а победный экран не открылся.

Худшее, что может сделать разработчик в такой ситуации — начать в панике переписывать всю программу наугад.

Сегодня мы освоим профессиональные методы отладки: заглянем «под капот» игры через консоль браузера, научимся раскрашивать состояния индикаторами и разберем системный алгоритм поиска багов».

Этап 2. Что такое баг и какие бывают (6 мин)

> 🖥️ **Слайд 2: Классификация багов**
>
> *Содержание слайда:* Таблица: критичные (игра не идёт) / важные (игра идёт, но плохо) / мелкие (косметические).

«Баг — это любое расхождение между тем, как игра должна работать, и как она работает на самом деле. Баги бывают разной критичности — и чинить их надо по приоритету».

Категория	Что ломается	Пример	Приоритет
Критичный	Игра не запускается / нельзя пройти	NPC не даёт квест, игра застревает	Чинить первым
Важный	Игрок может пройти, но опыт портится	NPC реагирует неверно; подсказки или сюжетные включения противоречат задумке автора, не ясно, куда двигаться на уровне	Чинить вторым
Мелкий	Косметика, опечатки, тайминги	Текст с опечатками; реплики рядом с объектами в игре мешаются; анимация сбоит и т.п.	Если останется время

> ⚠️ **Правило приоритета**
>
> Не чините мелочи, пока есть критичные баги. Это частая ошибка новичков: «текст красивее, а играть нельзя». Сначала — критичные, потом — важные, потом — мелкие.

Этап 3. Как искать баг: воспроизведение (6 мин)

> 🖥️ **Слайд 3: Шаг 1: воспроизвести баг**

«Баг нельзя починить, если его нельзя воспроизвести. Первый шаг — записать точную последовательность действий, которая приводит к багу. Если баг воспроизводится не всегда — это тоже информация».

Алгоритм поиска бага

1. Воспроизвести баг: записать шаги точно (что нажал, в каком порядке).
2. Проверить: воспроизводится ли каждый раз? Иногда? При определённых условиях?
3. Открыть консоль отладки «Урсулы» — там видны переходы между состояниями и ошибки.
4. Найти момент, где логика пошла не так: какой переход сработал не тот, какая переменная имела не то значение.
5. Сформулировать гипотезу: «Я думаю, проблема в том, что...»
6. Проверить гипотезу: внести одно изменение, воспроизвести баг снова.
7. Если исправилось — отлично. Если нет — вернуться к шагу 4.

> 💡 **Правило одного изменения**
>
> Меняй по одному параметру за раз. Если поменял 3 вещи и баг пропал — ты не знаешь, что именно помогло. И хуже: мог внести новый баг.

Этап 4. Консоль отладки «Урсулы» (5 мин)

> 🖥️ **Слайд 4: Что пишет консоль ?**

«Когда игра запущена в браузере, она непрерывно ведет внутренний бортовой журнал — консоль. Туда записывается каждый шаг: какое событие сработало, в какое состояние перешел объект, какие

переходы выполняются, какие переменные меняются, какие ошибки возникают».

Консоль отладки «Урсулы» — твой главный инструмент.

Как открыть консоль в браузере:

1. *Универсальный способ:* кликните правой кнопкой мыши в любом свободном месте страницы игры и выберите пункт меню **«Посмотреть код»** (или «Исследовать элемент»).
2. *Через настройки браузера:* откройте главное меню браузера (три точки или полоски в правом верхнем углу) → раздел «Дополнительные инструменты» (или «Разработка») → **«Инструменты разработчика»**.
3. В открывшейся панели выберите вкладку **Console** (Консоль).

Реальная строчка в консоли	**Что она означает**	**Что проверить разработчику**
Переменная "robot_mode" изменена с 0 на 1	Значение переменной обновилось в глобальной памяти базы	Сработала команда setVar из lnk или кнопка пульта
Evaluating condition: ... == 1 Условие ... оценено как: ИСТИНА	Движок проверил ограждающее условие на стрелке перехода	Условие выполнено, стрелка открыта для перехода (если ЛОЖЬ — переход блокируется)
[HSM_LOG] Вызван переход по событию ... из состояния А в Б	Автомат переключил состояние объекта	Проверить, совпадает ли целевое состояние с задуманным
[HSM_LOG] В состоянии выполнена команда: <Модуль.Команда>	Сработала конкретная команда на входе или по событию	Команда отдана исполнительной системе объекта

❌ [МодульДвижения] Нет позиции для отслеживания цели	Критическая ошибка: моб не может найти координаты объекта	Проверить имя целевого объекта в команде поиска (совпадает ли имя с моделью на карте)
⚠️ [МодульПеременных] Отслеживание уже активно...	Предупреждение: повторный вызов подписки на переменную	Не критично, но говорит о дублировании команды СледитьЗаПеременной

> ℹ️ **В Inky — Issues**
>
> В Inky тоже есть панель Issues: WARNING (loose ends, не используется) и ERROR (syntax error, unmatched divert). Перед интеграцией диалога в «Урсулу» — исправьте все ERROR.

Этап 5. Типичные баги и их причины (5 мин)

> 🖥️ **Слайд 5: Метод цветных маяков**
>
> *Содержание слайда:* Три блока схемы, окрашенные разными цветами: Синий (режим 0) → Зеленый (режим 1) → Красный (режим 2).

📣 **Действия учителя:**
«Консоль дает много информации в текстовом виде, но в 3D-мире мы можем видеть состояние автомата собственными глазами с помощью **МодуляИндикации**.

По аналогии с тем, как настроена индикация разных режимов у робота-инженера, можно настроить любого вашего NPC или объект со сложной диаграммой поведения.

На входе каждого ключевого состояния поставьте команду перекраски шара (и не забудьте в стартовом состоянии включить индикатор)
Как это помогает на сцене:

- Запустили игру: над объектом горит синий шар.
- совершили действие, которое должно вызвать переход в другое состояние — проверили, произошла ли смена индикации.

Например, с роботом-инженером: при старте горит синий шар, нажали команду на терминале: шар робота вспыхнул зеленым, но сам робот не поехал. *Быстрый вывод:* сигнал дошел, автомат перешел в режим патруля, ошибка не в переменной! Значит, нужно искать ошибку внутри блока патрулирования — возможно, забыли включить команду движения ДвигатьсяПоКоординатам(). Если же шар над роботом остался синим — сигнал про изменение переменной вообще не дошла до автомата, ошибка где-то в блоке, отвечающем за передачу переменной — в ПРИМС или в Ink.

Этап 6. Оптимизация логики (4 мин)

📣 Действия учителя:

«Чем сложнее логика — тем больше багов. Оптимизация — это упрощение. Если диаграмма из 10 состояний работает хаотично — попробуй сократить до 4. Часто 80% поведения можно выразить меньшим числом состояний».

- Объединяй похожие состояния. Если «Преследование 1» и «Преследование 2» отличаются на 1 деталь — это одно состояние с условием.
- Удаляй неиспользуемые. Если состояние не имеет входящих переходов (кроме entry) — оно не нужно.
- Разделяй ответственность. Если в одном состоянии 6 команд от 6 разных модулей — попробуй разбить на 2 состояния.
- Используй иерархию. Глобальные события — на уровне родителя, не дублируй в каждом дочернем.

Этап 7. Практика: работа по баг-листу (10 мин)

> 🖥️ **Слайд 6: Чек-лист системной отладки**
> *Содержание слайда:* Алгоритм работы: ① Воспроизвести баг; ② Проверить консоль; ③ Сформулировать гипотезу; ④ Внести одну правку; ⑤ Перепроверить.

📣 **Действия учителя:**

«Каждый из вас открывает свой проект. Проверьте все режимы работы по нашему чек-листу. Если что-то не работает — используйте **правило одной правки**: измените один параметр, сохраните, запустите и проверьте. Не меняйте пять вещей сразу!».

Пошаговый план работы для учеников:

1. Откройте консоль браузера в окне игры.
2. Проверьте реакцию объекта, который вызывает диалог из Ink: отдайте каждую из команд по очереди.
3. Убедитесь по логам консоли, что переменная меняет значения: 0→1→2→0.
4. Посмотрите на NPC: переключаются ли цвета маяка синхронно с командами терминала?
5. Если NPC застрял — найдите сбойный узел по карте поиска неисправностей из раздаточного материала.
6. Внесите точечное исправление в Кибериаде GE или в файле .ink, сохраните и проведите повторный тест.

> 💡 **Если баг не воспроизводится**
>
> Иногда баг «сам проходит» — это тоже информация. Возможно, вы что-то изменили ранее и это починило баг. Запишите: «не воспроизводится после <изменения>».

Этап 8. Подведение итогов (2 мин)

> 🖥️ **Слайд 7: Что должно быть готово**
>
> *Содержание слайда:* Чек-лист: ✅ баг-лист с приоритетами ✅ ≥3 бага исправлено ✅ проверено воспроизведением.

Обсуждение:

Какой баг сегодня оказался самым коварным?

Как консоль помогла найти ошибку быстрее, чем угадывание наугад?

📣 **Действия учителя:**

«Сегодня вы освоили настоящий инженерный подход. Разработчик отличается от новичка не тем, что не делает ошибок, а тем, что умеет быстро и спокойно находить их причину с помощью инструментов отладки. На следующем уроке мы подготовим проекты к публикации!».

> ℹ️ **Домашнее задание**
>
> Доработать баг-лист: исправить ошибки. Подготовить проект к публикации — тема следующего урока.

📄 Раздаточный материал

Шаблон баг-листа

№	Симптом	Шаги воспроизведения	Категория	Источник (после дебага)	Статус
1			Критичный		
2			Важный		
3			Важный		
4			Мелкий		
5			Мелкий		

Чек-лист «Я нашёл баг»

- ☐ Баг воспроизводится (записаны шаги)
- ☐ Определена категория (критичный/важный/мелкий)
- ☐ Открыта консоль отладки, виден источник
- ☐ Сформулирована гипотеза причины

- ☐ Внесено ОДНО изменение
- ☐ Баг перепроверен воспроизведением
- ☐ Если исправлено — статус «done». Если нет — новая гипотеза.

Памятка по чтению логов в консоли:

Каждая служебная строка состоит из трех частей:

[Источник лога] [Имя_модели :: ID_на_сцене | Активное_состояние] Сообщение

- **[HSM_LOG]** — записи самого автомата состояний (смена состояний, вход/выход, проверка условий).
- **[Иконка Модуля]** — отчеты функциональных систем (например, 🏃 МодульДвижения, 🎬 МодульАнимаций, ⏱️ Таймер).
- **Зеленая цепочка перехода:** Условия для перехода выполнены → Выполнен выход из состояния → Выполнен вход в состояние.

  Если эта цепочка прерывается словом ЛОЖЬ — переход не сработал из-за условия.
- **Поиск ошибок:** строки со значком ❌ подсвечиваются красным фоном и прямо указывают причину сбоя (например, отсутствие цели для движения).

🎓 Методические рекомендации учителю

Урок развивает инженерное мышление: баги — это нормально, их надо искать системно. Главная цель урока — преодолеть хаотичные действия учеников при возникновении ошибок. В средней школе дети часто начинают удалять случайные блоки или пересобирать граф заново. Задача наставника — приучить ребят к аналитическому алгоритму: симптом → консоль → гипотеза → точечная проверка.

Важно, чтобы ученики поняли: отладка — это отдельный навык, не менее важный, чем написание логики.

💬 **Скрипт-помощник для работы с группами**

Ситуация	Фраза учителя
Ученик не понимает, с чего начать поиск бага	*«Пройди игру до бага. Открой консоль. Пройди ещё раз. Смотри, что пишется в момент бага — скорее всего там подсказка».*
Ученик хочет переписать всю диаграмму	*«Стоп. Не переписывай. Найди конкретный переход, который срабатывает не так. Меняй только его».*
Ученик расстроен, что багов много	*«Баги есть у всех, даже у профессионалов. Возьми 3 самых критичных — остальное на потом. Это нормальный процесс».*
Ученик поменял сразу три вещи в схеме и запутался еще сильнее	*«Откати изменения назад кнопками Отменить в левой панели Кибериады GE. Вноси только одну правку за раз, иначе ты сам не поймешь, что именно помогло, а что сломало проект».*

⚠️ Возможные трудности и решения

Ситуация	Решение
Ученик не может воспроизвести баг	Попросите описать, в какой момент он заметил странность. Попросите пройти медленно, шаг за шагом, называя действия вслух. Часто баг воспроизводится при определённой последовательности.
Ученик «чинит» и ломает другое	Напомните правило одного изменения. Если после правки появилось новое — откатите правку, подумайте ещё. Не наслаивайте изменения.
В консоли слишком много служебных записей, ребенок теряется	Научите ребят пользоваться полем фильтра (поиска) вверху консоли: • Введите **HSM_LOG** — консоль скроет весь мусор и покажет только чистую цепочку смены состояний и проверок условий. • Введите имя переменной (например, **robot_mode**) — консоль

Ситуация	Решение
	покажет только моменты ее переключения. • Введите символ ❌ или включите фильтр *Errors* — консоль покажет только критические ошибки.
Ученик пытается «оптимизировать» рабочую логику	Напомните правило: «работает — не трогай». Оптимизация нужна, когда есть баги или тормоза. Если всё работает — оставь.
Ошибок в консоли нет, но робот ведет себя неправильно	Это логическая ошибка в условиях. Используйте световую трассировку: посмотрите, какой цвет маяка горит над головой робота прямо сейчас. Цвет покажет, в каком именно блоке находится логика.

✅ Критерии оценки

✅ Успешно	❌ Требует доработки
• Баг-лист заполнен (≥5 пунктов) с приоритетами • ≥3 бага исправлены • Каждое исправление проверено воспроизведением • При возникновении бага ученик точно называет сбойный блок или переход. • Использовалась консоль отладки	• Баг-лист пустой или хаотичный • Баги «чинятся» без воспроизведения • Консоль не открывается, поиск ошибок ведется случайным угадыванием. • Правки вносятся массово, без проверки • Ученик не понимает, в каком состоянии находится логика в момент сбоя.

💡 Дополнительные советы

- Покажите на проекторе разбор одного бага — это научит методу.
- Если у ученика нет багов — скорее всего, он их не видит. Дайте поиграть в его игру соседу, попросите записать странности.
- Не оценивайте «сколько багов исправлено» — оценивайте метод: воспроизведение → консоль →

гипотеза → одно изменение → проверка.

🏠 Домашнее задание

Доработать баг-лист: исправить оставшиеся ошибки. Подготовить проект к публикации.