УРСУЛД

ПРИ ПОДДЕРЖКЕ
ФОНДА
ПРЕЗИДЕНТСКИХ
ГРАНТОВ

Сюжетное ветвление в играх (Ink и ПРИМС)

Создание диалогов и квестовых цепочек



Зачем сюжету ветвиться?

Какие бывают сюжеты

1







Ловушка «комбинаторного взрыва» в игровых сюжетах

Иллюзия бесконечной свободы

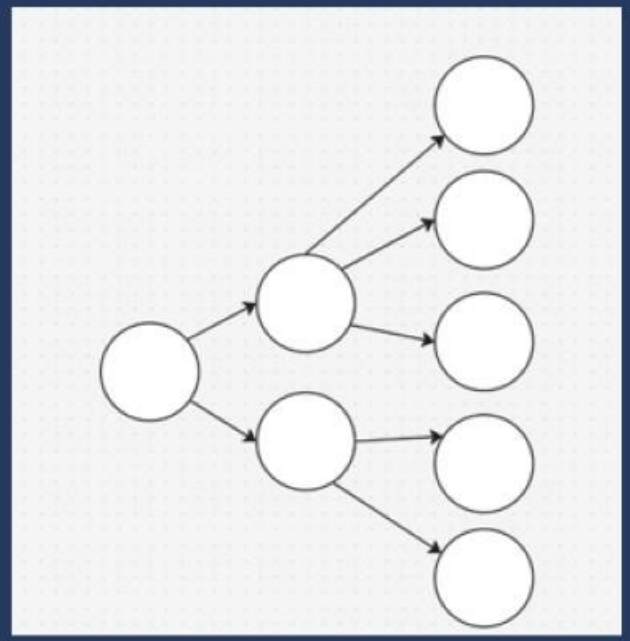
Начинающие авторы игр часто пытаются построить сюжет как дерево, где каждый выбор игрока порождает совершенно новую ветку событий.

Проблема комбинаторного взрыва

- Если на каждом шаге давать игроку выбор всего из 2 вариантов, то через 5 развилок разработчику придется написать уже $2^5 = 32$ отдельные сюжетные линии.
- 90% этого контента обычный игрок даже не увидит за одно прохождение.
- Для школьников 5–7 классов эта попытка заканчивается брошенным проектом: объем работы лавинообразно нарастает, в связях возникает путаница.

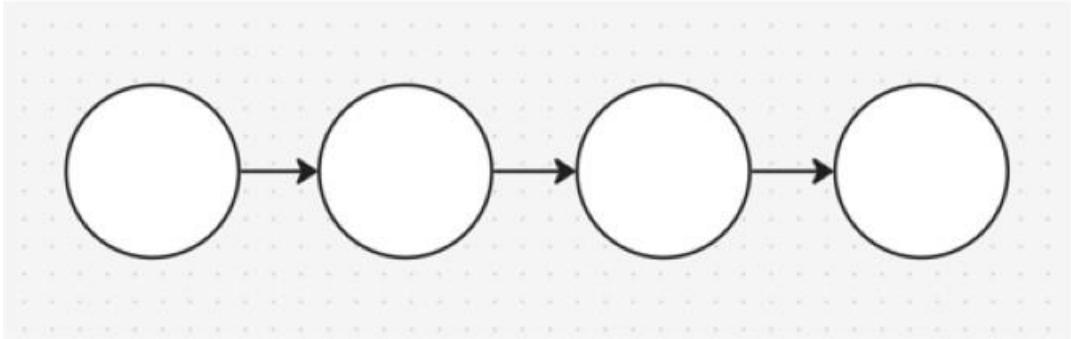
Инженерное решение: в геймдизайне свобода выбора создается не бесконечным умножением веток текста, а схемами управления сюжетом и работой с параметрами игрового мира (переменными и счетчиками).

Линейный сюжет



События разворачиваются строго последовательно:

Шаг А → Шаг Б → Шаг В



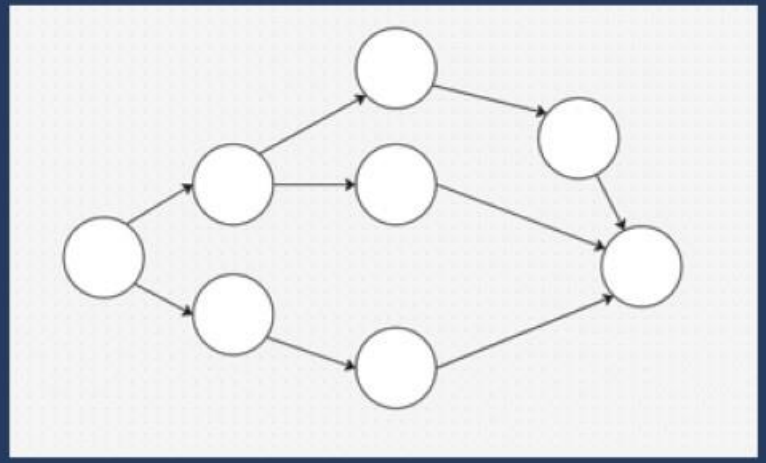
Выбора нет либо он косметический.

Плюсы: контроль над темпом истории, предельная простота настройки в конечном автомате.

Минусы: игрок чувствует себя пассивным зрителем, реиграбельность стремится к нулю.

схема «Алмаз»

ветвление собирается в «бутылочное горлышко»



Принцип

В ключевой точке сюжет разделяется на несколько параллельных вариантов решения задачи, но затем все пути гарантированно сходятся в единую обязательную контрольную точку.

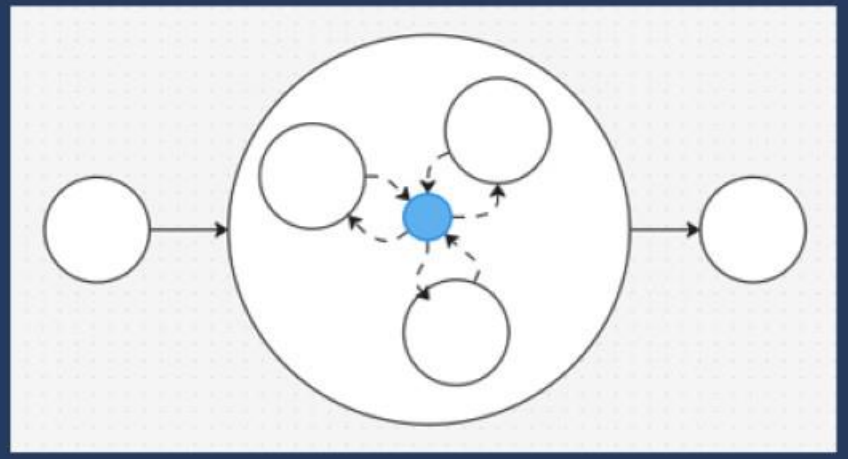
Пример: игрок-космонавт может проникнуть на базу по-разному:

- через взлом терминала,
- поиск обходного пути через скалы
- включение генератора.

Какой бы путь он ни выбрал, сюжет сходится в точке «Космонавт оказался внутри купола».

Преимущество: дает игроку ощущение свободы принятия решений, сохраняя при этом финал и ключевые этапы под полным контролем автора.

Схема «Сюжетный хаб»



Версия сюжета, в которой появляется персонаж-квестодатель (или обстоятельва), выдающий независимые задачи. Переход дальше происходит по какому-то триггеру.

Как устроена игровая петля хаба

- Фаза 1: брифинг. Герой обращается к хабу (узлу) и получает задачу.
- Фаза 2: свободное исследование. Игрок выполняет задачи в любом удобном ему порядке.
- Фаза 3: отчет и оценка. Игрок возвращается к хабу. Тот опрашивает параметры мира, дает обратную связь о текущем прогрессе и либо отправляет доделывать дела, либо переводит сюжет дальше.

Почему это удобно

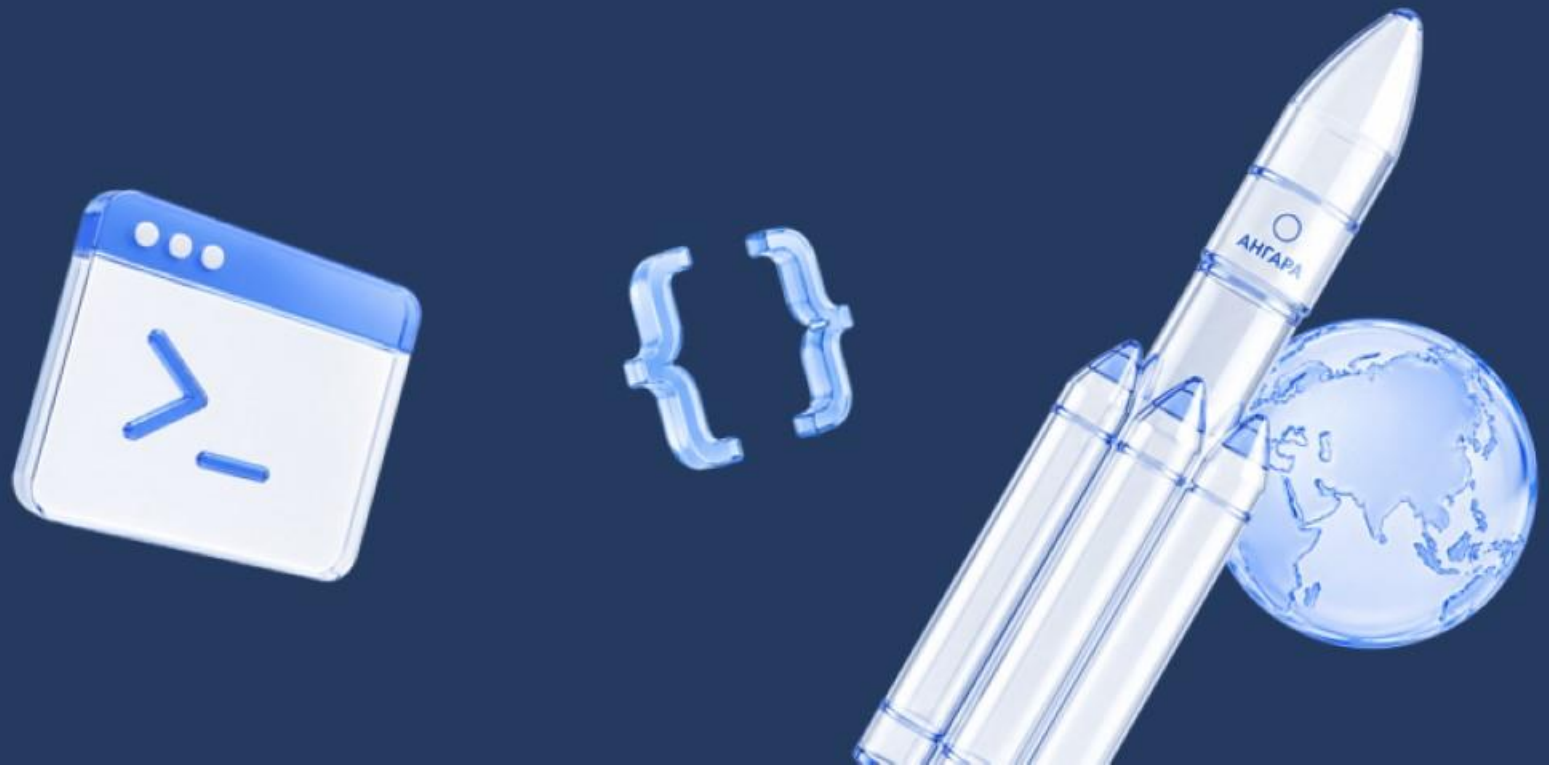
- Сценаристу не нужно описывать порядок выполнения задач.
- Вся логика проверки сосредоточена в одном объекте-хабе, а не «размазана» по всему уровню.

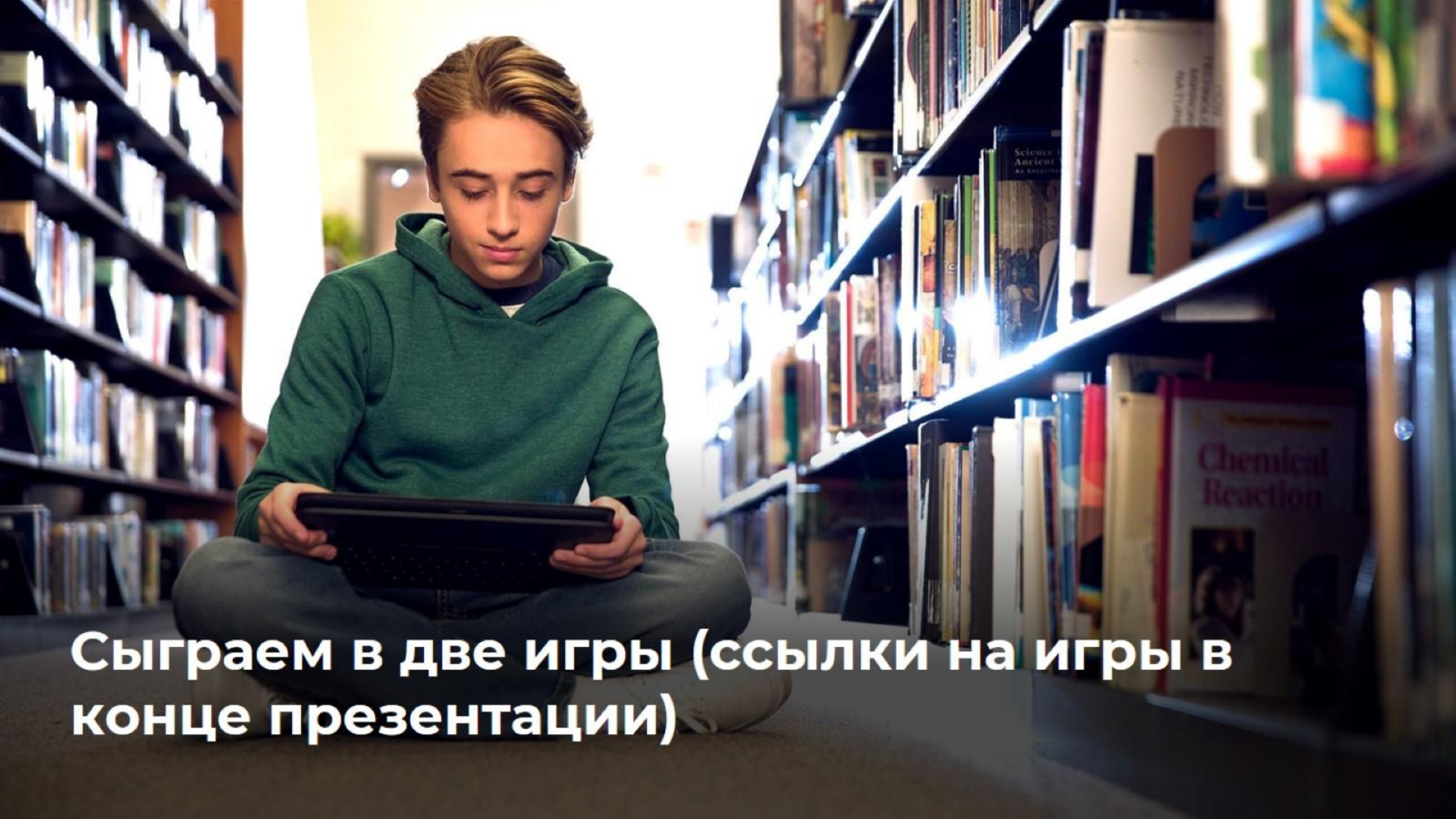
Посмотрим на примеры

Линейный квест (конкретный порядок)

Проверка выполнения условия (достаточно или нет)

2





Сыграем в две игры (ссылки на игры в конце презентации)

Линейная игра-квест

Суть квеста: изменение переменных по сюжетному порядку

Нужно последовательно взять яблоко, накормить ежа, взять ключ, отпереть ворота, кликнуть по флагу.

Три булевы переменные: has_apple, ezh_fed, has_key

Задача без линейной последовательности

Суть квеста: «сбор» ключевых объектов и проверка условий

На карте есть кнопки трех видов: зеленые, красные и синие, нужно нажать не менее 10 зеленых.

Нельзя нажимать на другие.

Три переменные: press_count (число), is_win (число), wrong_press (булева),

Сюжетные вариации шаблона

«Кнопочный квест»

Паттерн легко переодевается в любой игровой сеттинг:

Счетчик правильных действий
+ Флаг фатальной ошибки
+ Оценка прогресса в хабе

Инженерный квест («ремонт реактора/разминирование»):
Правильные объекты: исправные охлаждающие стержни или переключатели питания.
Ловушки: поврежденные модули (вызывают взрыв).

Экологический учет («сортировка мусора»):
Правильные объекты: перерабатываемый пластик .
Ловушки: неперерабатываемый пластик (всё насмарку!).

Алхимия или травничество («сбор ингредиентов»):
Правильные объекты: целебные травы.
Ловушки: похожие ядовитые растения (отравляют весь сбор).

Детективное расследование («поиск улик»):
Правильные объекты: улики на месте преступления (оружие, следы, отпечатки).
Ловушки: фальшивые улики, оставленные преступником.

Как мы это делаем?

Инструмент нарратива: использование языка Ink

Не столько программирование, сколько разметка текста

3



Ink: язык сценариев для ветвящихся историй

Ink – это открытый профессиональный язык разметки сценариев, созданный студией Inkle (на нем написано несколько крупных игр.

Для 3D-игры в «Урсуле» используется облегченный вариант этого языка.

Чем хорош Ink:

- Сценарий пишется в текстовом редакторе (лучше использовать редактор Inky) простым русским языком.
- Простые правила, и их не много.

Принцип разделения ответственности:

- В визуальном графе Кибериады GE мы настраиваем **физику и поведение**: где находится моб, куда он идет, открыта ли дверь, сколько очков здоровья осталось.
- В текстовом файле .ink мы пишем **диалоги и драму**: подсказки от мира, прямую речь, мысли героя, варианты ответов.

Продвинутые ученики могут использовать Ink для визуальных новелл (редактор на портале, понятный гайд)

Базовый синтаксис Ink

1. Точка входа (-> start, например)
Самая первая строка скрипта. Указывает, какой узел должен открыться при старте игры.

2. Объявление узлов (=== имя_узла ===)
Смысловой блок диалога. Имя пишется латиницей без пробелов (например, === robot_sos ===).

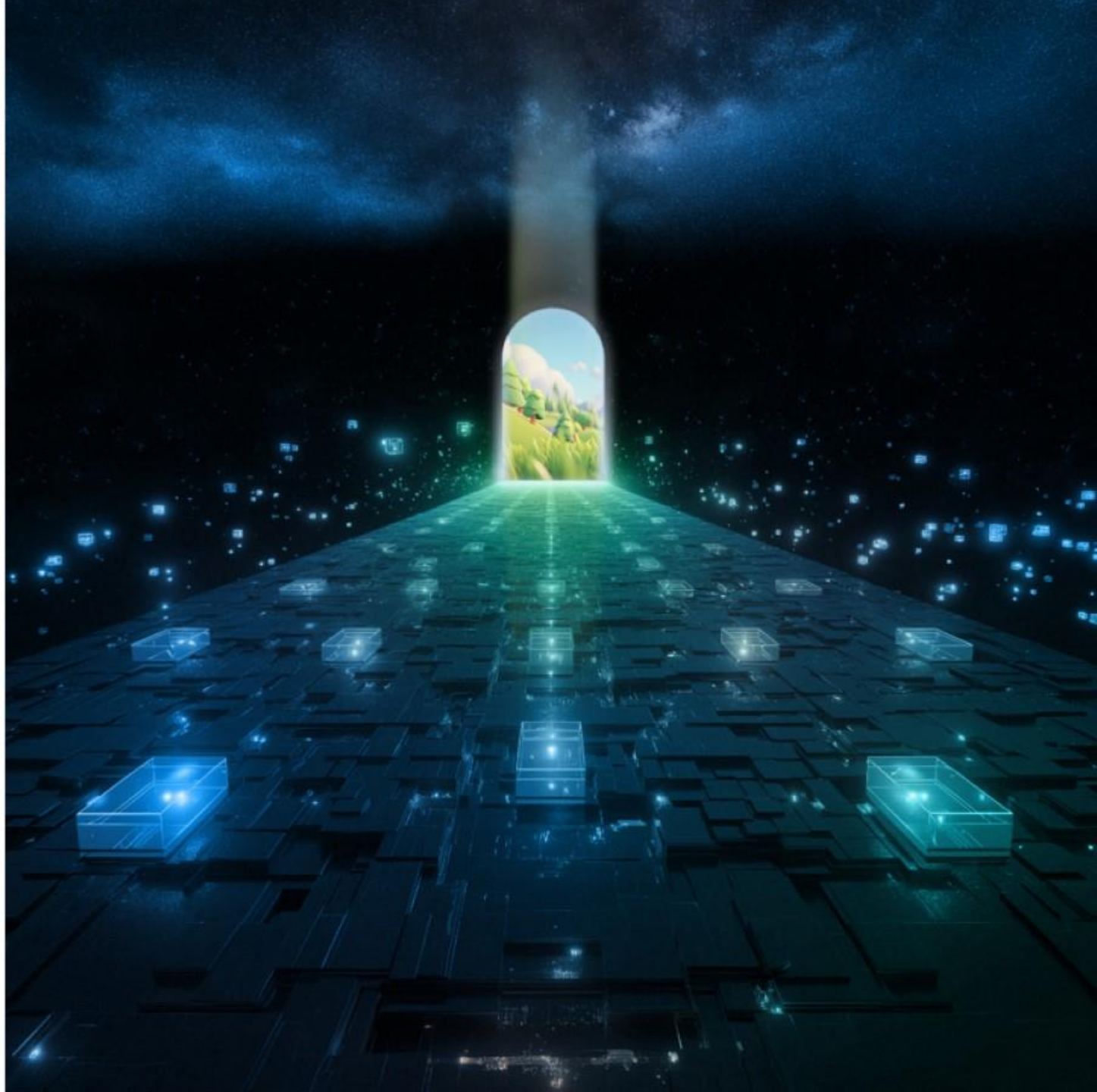
3. Текст реплики:
Обычный текст. Если реплика принадлежит персонажу, пишется в формате: Имя: Текст фразы.

4. Интерактивный выбор ответа (+ [Текст выбора]):
Символ плюса + и квадратные скобки [...] создают кликабельную строку в интерфейсе.

5. Переход к следующему узлу (-> имя_узла):
Стрелка -> перенаправляет игрока к следующему узлу сюжета после клика по выбранному ответу.

Посмотрим пример

- История линейная, в конце выбор
- Простой переход из «комнаты» в «комнату»
- Не для 3D (синтаксис немного другой)



Что еще можно?

Условия, переменные, функции

Ветвление реплик по условию { условие: ... - else: ... }
Позволяет персонажу говорить разное: в зависимости от состояния игры, не покидая текущего узла:

```
{ first_meeting:
    Робот: Приветствую, капитан! Обойдите базу и нажмите
зеленые кнопки.
    + [Приступаю к поиску!]
        ~ first_meeting = false
        -> DONE
- else:
    Робот: Рад видеть снова! Как успехи с поиском?
}
```

Многоступенчатая проверка переменной
Сценарий умеет проверять диапазоны чисел:

- press_count >= 13: → идеальный результат (найдены все);
- press_count >= 10: → достаточный результат (найдено большинство);
- press_count > 0: → промежуточный статус.

Интерполяция (вывод числа прямо в текст)
Синтаксис {имя_переменной} внутри текста автоматически подставляет актуальную цифру на экран:
В игре: «Робот: активировано недостаточно терминалов: всего 5 штук».

Движение по сценарию игры

В конце каждой ветки диалога нужно поставить команду перехода.

1. **Реплика + [Ответ игрока] и Команда -> имя_узла (переход к следующему блоку повествования)**

```
=== init ===
Вы совершили посадку на исследовательской планете. Системы
    связи челнока повреждены при входе в атмосферу...
+ [Открыть бортовой журнал] -> open_journal
+ [Осмотреться] -> look_around
```

1. **Команда -> DONE (узел завершен, игра продолжается)**

```
=== open_journal ===
Запись в журнале: «Первым делом найдите дежурного робота
    станции. До разговора с ним не трогайте незнакомое
    оборудование!»
-> DONE
```

2. **Команда #ending -> END (финал истории):**

```
=== lose_section ===
Бортовой журнал: Миссия завершилась неудачей. Системы базы
    обесточены, восстановить питание станции не удалось...
~ setVar("is_win", 0)
#ending
-> END
```

Локальные переменные в Ink

Подготовка
сценария
к приему данных
из внешнего мира

Обязательное объявление переменных (VAR)
Прежде чем Ink сможет прочитать данные из игры, эти переменные должны быть задекларированы в самом начале скрипта с начальными значениями:

```
EXTERNAL setVar(name, value)
VAR press_count = 0
VAR wrong_press = false
VAR first_meeting = true
```

```
VAR has_apple = false
VAR ezh_fed = false
VAR has_key = false
```

Изменение переменных внутри Ink через тильду (~):
Любые вычисления или присваивания внутри текста сценария пишутся со знака тильды:

```
+ [Задача ясна, приступаю к поиску!]
    ~ first_meeting = false
    -> DONE
```

Как ПРИМС и Ink обмениваются данными

Двусторонний мост между

4







Два потока взаимодействия

Откуда куда направлять?

Где: в центральном диспетчере (StateController в Урсуле)

В памяти запущенной игры существует единый контроллер глобальных переменных. К нему имеют доступ как автоматы Кибериады GE, так и интерпретатор Ink.

Два потока информации:

- **Поток 1: «Мир → Сценарий» (ПРИМС → Ink)**
  Игрок нажимал кнопки на карте → переменные press_count и wrong_press изменились в памяти базы → при клике на робота/Ежа/ключ и другие объекты автомат принудительно «проталкивает» эти значения внутрь диалога Ink.
- **Поток 2: «Сценарий → Мир» (Ink → ПРИМС):**
  Внутри диалога сюжет дошел до развилки финала → скрипт Ink через служебную функцию выставляет значение переменной is_win в памяти игры → завершение сценария посылает сигнал в автомат робота, и тот переключает экран победы или поражения.

Где меняется/устанавливается переменная – оттуда и сообщаем!

МодульПеременных



Реакция на действия в мире (УстановитьЗначение, ПрибавитьЗначение, ОтнятьЗначение): когда игрок как-то взаимодействует с переменной (нажимает на правильную или неправильную кнопку, например), нужно прибавить значение в переменную счета кнопок или установить значение true для переменной, которая следит за нажатием неверной кнопки.

Слежка за изменениями в мире (СледитьЗаПеременной): когда игрок кликает на ключ и подбирает его), в сценарии меняется переменная has_key = true. Автомат тут же улавливает это событие (ЗначениеПеременнойИзменилось) и ворота (подписанные на эту переменную) готовы открываться по клику.

МодульПовествования



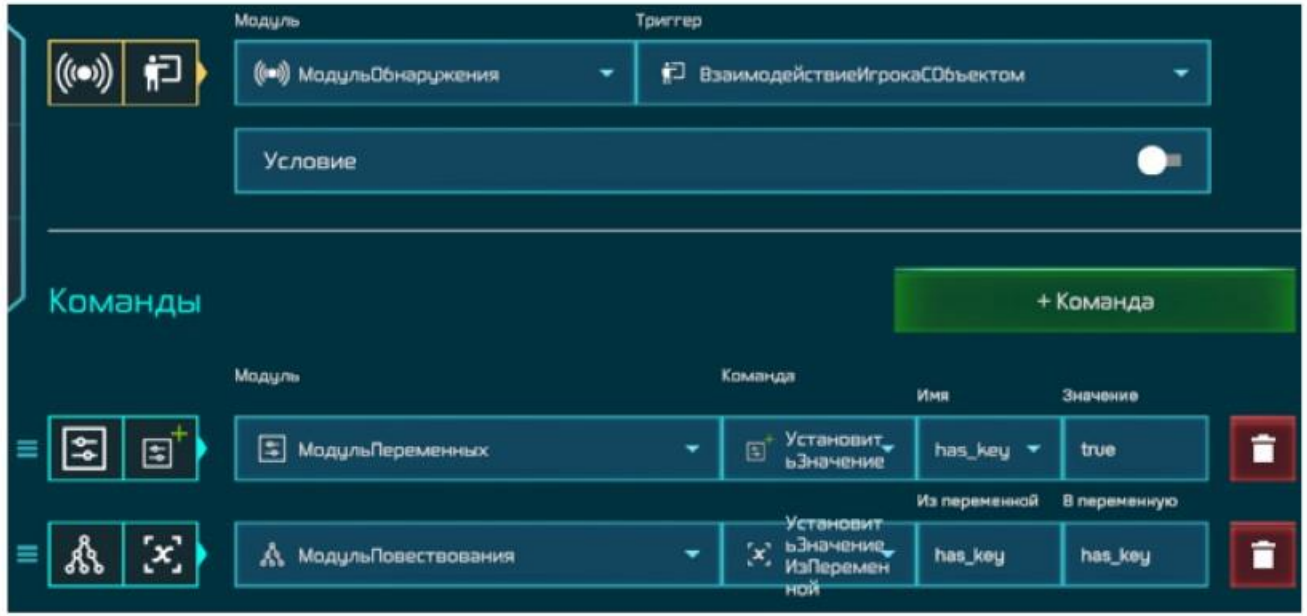
Старт истории с нужного места (ПерейтиКУзлу): мы указываем, с какой реплики начать общение (например, узел robot или apple_green).

Связь между переменными в игре и в Ink (УстановитьЗначениеИзПеременной): копирует данные из глобальной переменной в редакторе в переменную в сценарии в Ink, чтобы Ink это учитывал при переходе в узел.

Завершение истории (ПовествованиеЗавершено): конец игры.

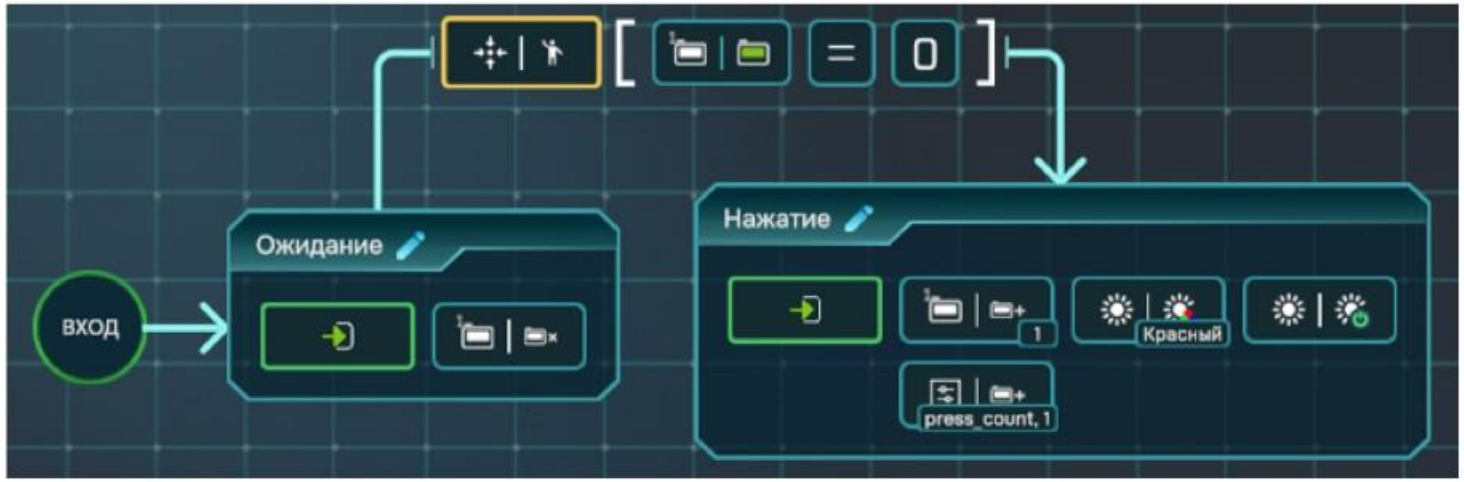
Разбор ПРИМС-диаграмм игры про Ежа

1. Диаграмма Ежа записывает значение ezh_fed из переменной в игре в переменную в Ink, и показывает верный текст для сытого Ежа (сразу по переходу в состояние).
2. Диаграмма ключа записывает значение has_key из переменной в игре в переменную в Ink, и показывает текст об открытии ворот (когда игрок провзаимодействует с воротами).
3. И Еж, и ворота подписаны на отслеживание «своих» переменных.



Разбор ПРИМС-диаграмм кнопок

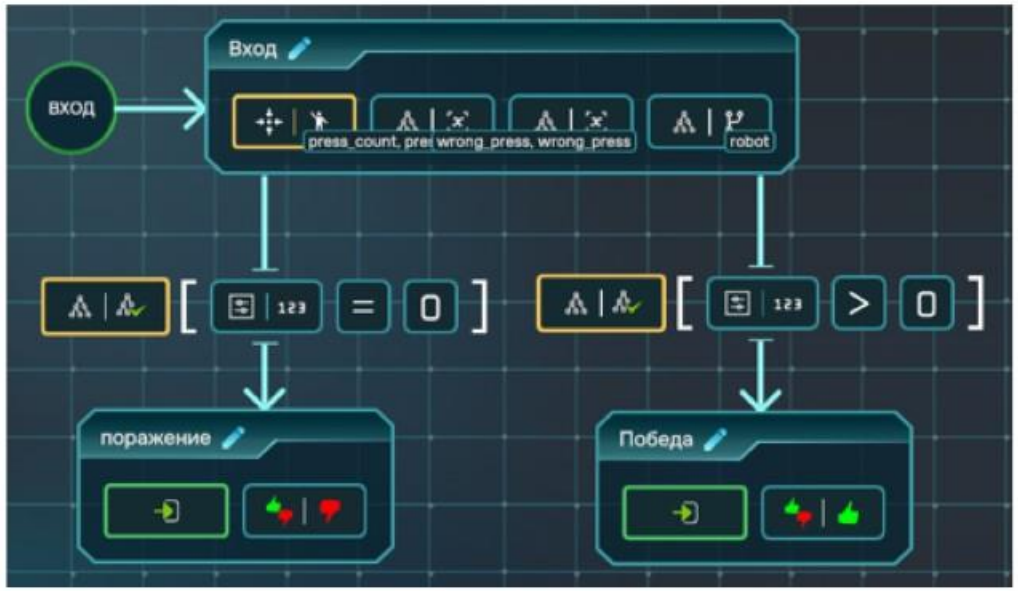
1. Верная кнопка добавляет значение в переменную press_count (кнопку можно нажать только раз)
2. Неверная кнопка устанавливает значение wrong_press в ИСТИНА.

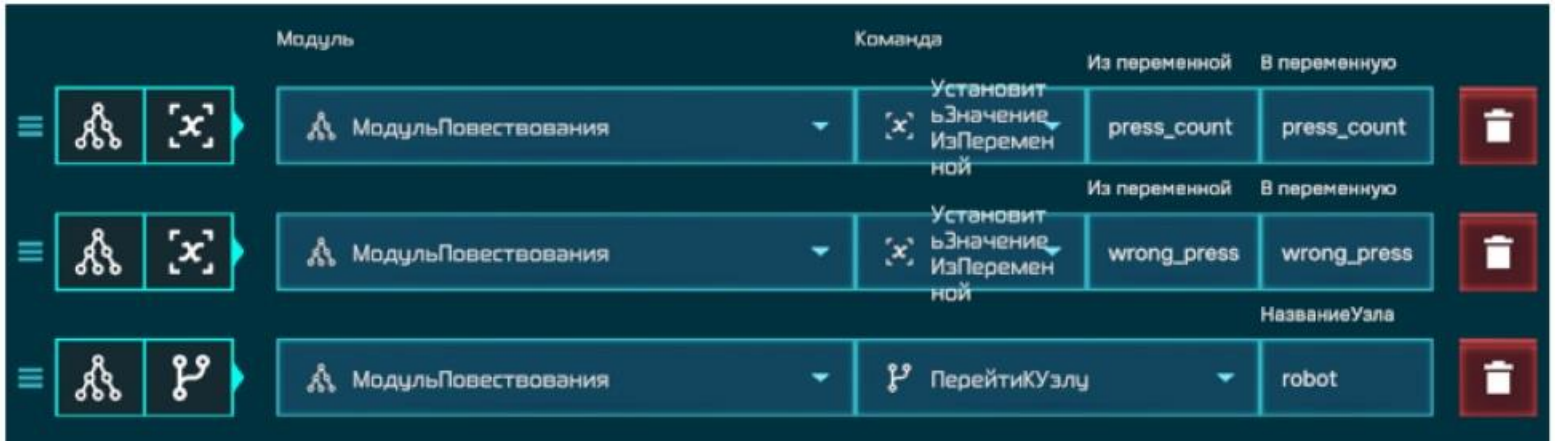




Разбор ПРИМС-диаграммы Робота

1. При взаимодействии СПЕРВА отправляет текущие значения переменных в Ink, а затем проигрывает узел robot в Ink.
2. Если завершилось повествование – проверяет условие победы





Поток 2: Сценарий -> Мир

функция EXTERNAL setVar

```
=== lose_section ===
Бортовой журнал: Миссия завершилась неудачей. Системы базы
    обесточены, восстановить питание станции не удалось...
~ setVar("is_win", 0)
#ending
-> END
```



Передача решений из диалога в игру:

Обычный скрипт Ink изолирован от игрового движка. Чтобы разрешить ему менять переменные игры, в самой первой строчке файла .ink декларируется внешняя функция: EXTERNAL setVar(name, value).

Как это работает:
Функция позволяет записывать глобальную переменную в StateController игры.

Синтаксис вызова функции внутри сценария:
пишется со знака тильды ~:
~ setVar("is_win", 1) или ~ setVar("is_win", 0)

Событие-триггер ПовествованиеЗавершено:
Это системное событие МодуляПовествования. Оно генерируется в автомат **только тогда, когда Ink дошел до меток #ending -> END**.

Переменную с точным именем is_win (тип number, стартовое значение 0) необходимо заранее создать в настройках сцены Урсулы.

Методические дополнения

Самые распространенные проблемы, отладка кода и тестирование

5



Чек-лист для быстрой проверки

1. **Настройки сцены в Урсуле (Глобальные переменные):**
   Созданы ли в проекте переменные с точными именами: press_count (number, 0), wrong_press (boolean, false), is_win (number, 0)? Если переменная забыта — игра не сможет в нее записать значение.
2. **Наличие файла сценария:**
   Загружен ли файл .ink в поле «Сценарий» в настройках сцены Урсулы?
3. **Точное совпадение имен узлов (регистр важен!):**
   Проверьте строку в скобках команды ПерейтиКУзлу(...) в Кибериаде GE и заголовок узла в Ink (после ===). Имя robot и имя Robot — разные!
4. **Первые строки файла Ink:**
   Объявлены ли все переменные, которые задействованы в сценарии Ink?
5. **Порядок команд внутри ноды:**
   Команды УстановитьЗначениеИзПеременной должны стоять **выше (т.е. случиться раньше)** команды ПерейтиКУзлу.

Типичные ошибки в синтаксисе

Использование -> DONE вместо -> END в финале
Настройка сцены будет выдавать ошибку записи в конфиг.

Разные опечатки в вызове setVar

Игрок будет всегда проигрывать.
Если написать ~ setVar("is_win ", 1) (с лишним пробелом внутри кавычек) или ~ setVar("is_Win", 1) (с заглавной буквы), то переменная is_win в игре всегда останется равной 0, и сработает условие поражения [is_win == 0].

Культура отладки

Тестировать по частям, а не всё сразу:
Поставили робота и 1 зеленую кнопку → проверили, что счетчик стал равен 1 и робот видит эту единицу → поставили красную кнопку → проверили, что робот ругается при ошибке → размножили остальные кнопки на сцене.

Временные «читерские» условия для быстрой проверки:
Чтобы протестировать победу, не нужно каждый раз нажимать все 13 кнопок по всей карте. На этапе настройки в скрипте Ink можно временно написать условие - press_count >= 1: для победы. Проверили, что победа достигается при нажатии одной кнопки → вернули число 13.

Цветовая индикация как маркер для состояний:
Напомните ученикам окрашивать состояния в сложных или комплексных программах.

Материалы показанных игр

(ПРИМС-диаграммы и Ink-файлы)

Игра «Голодный страж» (для десктопа):
https://games.kruzhok.org/games/2686?build=production

Игра «Кнопочный квест» (для десктопа):
https://games.kruzhok.org/games/3056?build=production

Ссылка на материалы обеих игр:
(Яндекс.Диск)

Материалы для углубленного изучения Ink

(и просто игры на Ink)

Курс-руководство на портале Урсула про Ink.



ВАЖНО: синтаксис Ink в 3D-играх на Урсуле немного отличается, это руководство предназначено для написания визуальных новелл!

Библиотека игр на портале Урсула, разделы

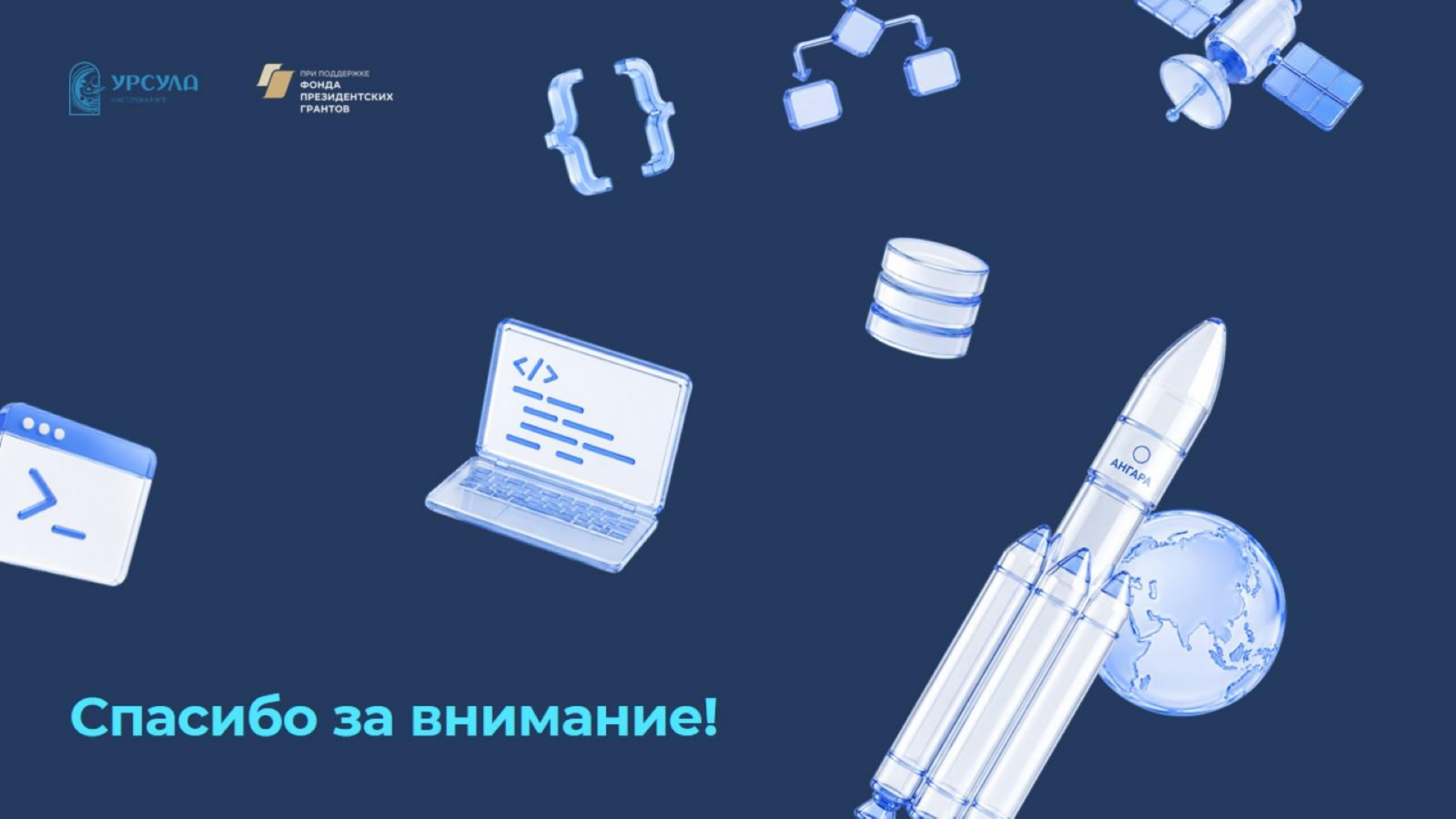
«Визуальные новеллы» «Конкурс интерактивных историй»





УРСУЛА

ПРИ ПОДДЕРЖКЕ ФОНДА ПРЕЗИДЕНТСКИХ ГРАНТОВ



Спасибо за внимание!