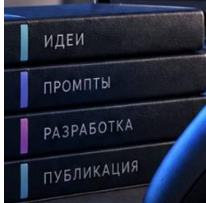
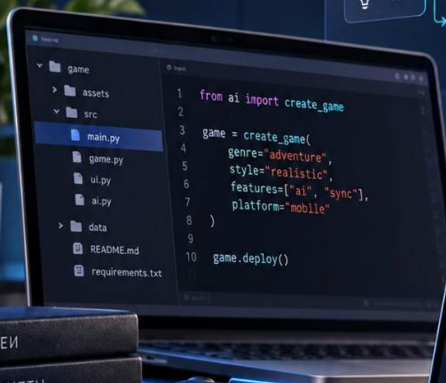
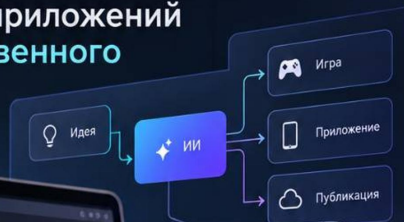


ВАЙБКОДИНГ

Практический самоучитель
по созданию игр и приложений
с помощью искусственного
интеллекта



Иван Зорин

Иван Зорин

Вайбкодинг

<https://litres.ru/74530072>

SelfPub; 2026

Аннотация

Создавать собственные игры и приложения теперь можно ещё до того, как вы научились писать код с нуля. Современные нейросети умеют собирать рабочие прототипы, изменять целые проекты, находить ошибки и помогать с графикой, интерфейсами и игровой логикой — но получить первую работающую версию гораздо проще, чем довести её до законченного продукта.

Эта книга — практический самоучитель по вайбкодингу для начинающих. Вы разберётесь, что сегодня умеют ChatGPT и Codex, Grok, Gemini, Claude и другие ИИ-инструменты, создадите первую браузерную игру, а затем перейдёте к настоящим проектам в GDevelop, Godot и Unity. Отдельные главы посвящены веб- и мобильным приложениям, генерации графики и звука, работе с Git, отладке, тестированию, безопасности и публикации готового проекта.

Содержание

Предисловие	4
Часть I. Что сейчас можно сделать внутри нейросети	7
Глава 1. Три разных вида вайбкодинга	7
Глава 2. ChatGPT и Codex: от разговора к проекту	10
Глава 3. Grok Build: приложение или игра внутри разговора	12
Глава 4. Gemini Canvas: уже не только код в чате	14
Глава 5. Claude Artifacts: прототипы отдельных механик	16
Глава 6. Replit Agent: приложение сразу в облачной среде	18
Глава 7. Как выбрать инструмент для первой идеи	20
Часть II. Основы работы с проектом	22
Глава 8. Папка проекта и зачем её понимать	22
Глава 9. Как ставить задачу, чтобы не переделывать весь проект	24
Конец ознакомительного фрагмента.	25

Иван Зорин

Вайбкодинг

Предисловие

Вайбкодинг часто показывают как почти комический способ программирования: человек пишет нейросети что-нибудь вроде «сделай игру про космический корабль», получает несколько сотен строк неизвестного ему кода, нажимает кнопку запуска и радуется тому, что корабль действительно летает. Для короткой демонстрации этого достаточно, однако человек, который попробует таким способом сделать хотя бы небольшую законченную игру, довольно быстро обнаружит, что настоящее занятие начинается после первого удачного запуска. Нужно поменять управление, добавить меню, исправить столкновения, сохранить прогресс, сделать нормальную работу на телефоне, заменить временную графику, разобраться с зависимостями и через две недели каким-то образом не разрушить всё то, что работало вчера.

Поэтому эта книга устроена как практический самоучитель, а не как сборник эффектных запросов. Здесь мы будем пользоваться нейросетями постоянно, иногда поручая им почти всю механическую работу, но проект при этом останется понятным человеку, который его создаёт. Чита-

телю не придётся сначала несколько месяцев изучать язык программирования и только потом переходить к собственной игре, однако основные понятия всё равно постепенно появятся: файл проекта, сцена, объект, компонент, событие, переменная, функция, состояние, игровой цикл, база данных, клиентская и серверная часть. Они будут вводиться тогда, когда без них становится трудно решить конкретную задачу, поэтому теория будет связана с тем, что уже находится перед глазами.

За последние два года изменился сам набор инструментов. Ещё недавно было разумно говорить, что обычная нейросеть только выдаёт код, который затем приходится самостоятельно переносить в редактор, однако на момент подготовки книги, осенью 2026 года, эта граница почти исчезла. Существуют среды, которые создают работающий веб-проект прямо внутри разговора, программные агенты, способные читать и менять полноценный репозиторий, игровые движки со встроенными ИИ-помощниками и сервисы, которые позволяют прототипировать приложение без локальной установки среды разработки. Именно поэтому в книге не будет одной «правильной» нейросети. Мы будем выбирать инструмент под задачу.

ChatGPT и Codex удобны, когда нужен разговор о проекте и затем работа с файлами и репозиторием; Grok Build и похожие среды позволяют быстро получить интерактивный проект внутри самой ИИ-среды; Gemini Canvas и Claude

Artifacts подходят для небольших веб-приложений, инструментов и прототипов; Replit Agent удобен, когда нужно сразу получить работающий облачный проект. Для игр мы отдельно разберём GDevelop, Godot и Unity, потому что эти движки позволяют пройти путь от почти полного отсутствия навыков до настоящего проекта, при этом каждый показывает программирование с другой стороны.

Главная задача книги состоит не в том, чтобы научить писать особенно хитрые промпты. Формулировка запроса имеет значение, но гораздо важнее приобрести привычку работать с программой как с системой, которую можно исследовать, менять и проверять. Если после прохождения книги вы сможете открыть незнакомый проект, понять его основные части, объяснить нейросети требуемое изменение, проверить созданный ею код и самостоятельно определить следующий шаг, значит обучение выполнило свою задачу.

Часть I. Что сейчас можно сделать внутри нейросети

Глава 1. Три разных вида вайбкодинга

Когда человек говорит, что «сделал приложение с помощью нейросети», за этой фразой могут скрываться совершенно разные процессы. В первом случае модель выдала ему один HTML-файл, который он сохранил и открыл в браузере. Во втором она создала несколько файлов приложения или готовую папку проекта, после чего работа продолжилась во внешнем редакторе. В третьем программный агент получил доступ к репозиторию, самостоятельно прочитал существующий код, изменил нужные файлы, запустил команды, увидел ошибки и исправил их. С точки зрения пользователя все три варианта похожи, потому что главным интерфейсом остаётся обычный язык, однако их возможности и риски различаются.

Самый простой уровень можно назвать генерацией автономного результата. Для небольшой браузерной игры иногда достаточно одного `index.html`, внутри которого находятся разметка, стили и JavaScript. Такой проект легко отправить

другому человеку, быстро открыть и так же быстро выбросить, если идея оказалась неудачной. Именно поэтому однофайловые прототипы полезны не только новичку: опытный разработчик тоже может проверить механику быстрее, чем создаст полноценный проект в движке.

Следующий уровень — многофайловый проект. Современное приложение обычно содержит десятки файлов, даже если пользователь видит всего несколько экранов. Отдельно хранятся компоненты интерфейса, стили, настройки, зависимости, изображения и серверная логика. Игровые движки тоже используют свою структуру. Здесь простой обмен кусками кода становится неудобным, поэтому нужен инструмент, способный создавать или менять целую папку.

Третий уровень — агентная разработка. Агент не ограничивается текстом ответа, а выполняет действия: читает проект, создаёт файлы, запускает тесты, вызывает сборку, смотрит ошибку терминала и повторяет цикл. Именно этот вариант ближе всего к тому, что обычно имеют в виду под зрелым вайбкодингом. Человек ставит задачу на уровне поведения продукта, а агент занимается значительной частью механического перевода этого требования в код.

Полезно уметь пользоваться всеми тремя режимами. Однофайловая генерация хороша для идеи, которую хочется увидеть сегодня; готовая папка проекта нужна, если результат должен жить в Godot, GDevelop или другом движке; агентная работа становится особенно ценной после того, как

проект перестал помещаться в одном разговоре и количество связанных файлов выросло.

Глава 2. ChatGPT и Codex: от разговора к проекту

Обычный разговор с ChatGPT хорошо подходит для проектирования. Здесь удобно придумать механику игры, разложить идею приложения на экраны, составить техническое задание, разобрать сообщение об ошибке или попросить объяснить непонятный участок кода. Если среда позволяет создавать файлы, можно получить готовый набор исходников или архив проекта, но при длительной разработке удобнее переходить к инструменту, который работает непосредственно с репозиторием.

Codex следует воспринимать не как «чат, который пишет побольше кода», а как программного агента. Его ценность проявляется тогда, когда задача затрагивает несколько файлов и требует понимания существующей структуры. Вместо того чтобы вручную копировать `Player.gd`, `Enemy.gd`, сцену и конфигурацию, можно открыть репозиторий и поставить задачу на уровне проекта: добавить систему здоровья, не менять управление, использовать уже существующий HUD и проверить сценарий перезапуска. Агент сначала изучает код, затем вносит изменения там, где они действительно нужны.

Для первого опыта программный агент вообще не обязателен. Полезно начать с автономной браузерной игры и попросить создать её в одном файле без внешних библиотек.

Например, игрок управляет кругом, уклоняется от препятствий и набирает очки, а после столкновения получает кнопку повторного запуска. Важно сразу попросить нормальную работу с клавиатурой и, если вы хотите открыть игру на телефоне, простое сенсорное управление. Чем меньше зависимостей у первой версии, тем легче понять, почему что-то не запускается.

Когда однофайловый прототип становится неудобным, попросите разделить его на несколько файлов, но явно запретите менять игровую механику. Это полезное упражнение: вы увидите, что одна и та же программа может быть организована по-разному, а рефакторинг не должен автоматически означать изменение поведения.

С игровыми движками агента можно использовать ещё интереснее. Godot хранит значительную часть проекта в текстовых форматах, поэтому модель способна создать `project.godot`, текстовые сцены и `GDScript`-файлы. Unity требует большей осторожности из-за структуры проекта и импортируемых метаданных, поэтому для новичка обычно надёжнее сначала создать пустой Unity-проект в самом редакторе, а затем позволить агенту добавлять скрипты и работать с уже существующей структурой.

Глава 3. Grok Build: приложение или игра внутри разговора

Среды вроде Grok Build показывают другой тип вайбкодинга: пользователь описывает приложение или игру и сразу получает работающую версию внутри самого сервиса. Такой подход особенно хорош для первой проверки идеи, потому что между запросом и реальным взаимодействием почти нет промежуточных шагов. Не нужно создавать папку, ставить зависимости и запускать локальный сервер только для того, чтобы выяснить, что выбранное управление неудобно.

Для первой пробы лучше выбрать проект с коротким и законченным игровым циклом. Например, персонаж перемещается по комнате, собирает десять предметов и избегает противника. После десятого предмета появляется победа, после столкновения с врагом — поражение. Если такая версия работает, следующим запросом можно добавить поддержку телефона, затем усложнить противника, а уже после этого заниматься визуальным стилем и хранением рекордов.

Главное достоинство интерактивной генерации заключается в скорости обратной связи. Фраза «персонаж слишком медленный» появляется не из абстрактного чтения кода, а после реальной попытки играть. Если кнопка на мобильном экране оказывается неудобной, это видно сразу. Поэтому такие среды полезно рассматривать не только как конструкторы

ры, но и как лаборатории, где проверяется сама идея продукта.

Если сервис позволяет экспортировать проект в репозиторий, это хороший момент для перехода к более обычной разработке. Прототип, который уже доказал, что в нём есть смысл, можно сохранить, подключить Git и продолжать в редакторе. При этом не следует путать удачный прототип с готовым коммерческим продуктом: пограничные случаи, работа на разных экранах, безопасность, хранение данных и длинные игровые сессии всё равно требуют отдельной проверки.

Глава 4. Gemini Canvas: уже не только код в чате

Представление о Gemini как о модели, которая способна только написать код сообщением, сегодня слишком узкое. Canvas позволяет создавать интерактивное приложение в отдельной рабочей области, видеть результат, изменять его через разговор и при необходимости смотреть код и сообщения об ошибках. Для небольшого веб-инструмента это означает, что человек может начать с описания поведения, а не с подготовки локального проекта.

Хорошими первыми задачами для Canvas являются калькулятор бюджета, трекер привычек, каталог личной коллекции, генератор карточек для обучения, викторина, редактор персонажа настольной игры или небольшая браузерная игра. В таких проектах интерфейс и логика достаточно ограничены, чтобы быстро увидеть результат, но уже есть состояние, пользовательский ввод и несколько сценариев поведения.

Если приложение должно хранить данные между посещениями, появляется следующая ступень сложности. Локальное хранилище браузера подходит для личного прототипа, но совместное приложение требует облачной базы и правил доступа. Когда среда автоматически подключает хранилище, важно всё равно понимать, какие данные доступны пользователям и кто имеет право их читать. Автоматическое созда-

ние инфраструктуры уменьшает количество ручной работы, но не отменяет необходимость задавать вопросы о безопасности.

Для Godot и Unity Gemini остаётся полезным как внешний технический помощник. Он может написать GDScript или C#, объяснить ошибку, предложить структуру сцены или разобрать систему сохранений, однако запуск полноценного редактора игрового движка остаётся отдельным этапом. Это хороший пример того, почему стоит различать генерацию работающего веб-приложения и работу с проектом в специализированном движке.

Глава 5. Claude Artifacts: прототипы отдельных механик

Claude Artifacts удобен тогда, когда результат должен появиться рядом с разговором и его можно сразу потрогать. Небольшие веб-инструменты, интерактивные визуализации, карточные мини-игры, редакторы данных и учебные приложения хорошо подходят для такого режима. Здесь полезно прототипировать не только целый продукт, но и отдельную механику будущей игры.

Предположим, вы хотите сделать карточную RPG в Godot, но пока не понимаете, интересна ли сама система боя. Нет смысла сразу строить меню, карту мира и сцены персонажей. Сначала можно попросить Artifacts создать маленький веб-прототип, в котором есть рука из пяти карт, здоровье игрока и противника, энергия и несколько эффектов. Через полчаса станет понятно, приятно ли выбирать карты, достаточно ли очевидны последствия действий и не скучно ли ждать следующего хода.

Если механика оказалась удачной, её правила переносятся в основной проект движка. При этом переносить буквально каждую строку веб-кода не обязательно. Гораздо ценнее сохранить спецификацию: какие существуют карты, когда заканчивается ход, как рассчитывается урон и каким образом игрок получает новые карты. После такого прототипа

техническая реализация в Godot или Unity становится значительно понятнее, потому что дизайн уже проверен.

Глава 6. Replit Agent: приложение сразу в облачной среде

Replit занимает промежуточное положение между разговорным ИИ и обычной средой разработки. Агент получает задачу, создаёт проект, запускает его, показывает результат и позволяет продолжать работу без отдельной локальной установки. Для веб-приложений это особенно удобно, потому что сервер, зависимости и среда выполнения уже находятся рядом с проектом.

Первые приложения здесь лучше строить вокруг одного законченного сценария. Если это список покупок, пользователь должен создать список, добавить позицию, отметить её купленной и после перезагрузки увидеть прежнее состояние. Если это дневник тренировок, сначала достаточно добавить тренировку и просмотреть историю. Регистрация, совместный доступ, фотографии и подписка могут появиться потом, когда основной сценарий не вызывает сомнений.

В облачной среде быстро становится видно различие между красивым прототипом и настоящим приложением. Страница может выглядеть убедительно, но данные пропадут после перезапуска сервера, форма не обработает ошибку, а два пользователя увидят одну и ту же запись. Именно здесь полезно впервые начать думать о сервере, базе данных и авторизации не как о сложной теории, а как о конкретных частях

работающего продукта.

Глава 7. Как выбрать инструмент для первой идеи

Лучше начинать не с вопроса о том, какая нейросеть «самая умная», а с того, каким должен быть результат через несколько дней. Небольшая браузерная игра или интерактивный инструмент хорошо подходят для Grok Build, Gemini Canvas или Claude Artifacts. Веб-сервис с базой, сервером и дальнейшим развитием удобнее строить в Replit или через полноценного кодового агента. Существующий репозиторий лучше отдавать инструменту, который действительно умеет читать и менять файлы, а не обычному чату с постоянным копированием фрагментов.

Для игр выбор дополнительно зависит от того, где проект должен жить после первых экспериментов. GDevelop подходит человеку, который хочет очень быстро увидеть игровую логику и не начинать с синтаксиса языка. Godot удобен для постепенного перехода к настоящему коду и хорошо дружит с программными агентами благодаря текстовым проектным файлам. Unity требует больше внимания к редактору и структуре, но даёт огромную экосистему, серьёзные 2D- и 3D-инструменты и множество готовых ресурсов.

Ничто не мешает сочетать сервисы. Идею можно обсудить с ChatGPT, отдельную боевую механику проверить в Artifact, затем собрать игру в Godot и использовать Codex для работы

с репозиторием. Инструменты будут меняться быстрее, чем проект. Поэтому долговечным навыком остаётся не привязанность к конкретному бренду, а способность описать поведение и проверить результат.

Часть II. Основы работы с проектом

Глава 8. Папка проекта и зачем её понимать

Начинающему исходный код часто представляется огромным текстом, который где-то внутри компьютера превращается в программу. Реальный проект больше похож на папку с взаимосвязанными документами. Один файл содержит настройки, другой описывает экран или сцену, третий отвечает за конкретную механику, рядом лежат изображения, музыка и данные. Нейросеть может создавать эти файлы за вас, но это не отменяет пользы от понимания их назначения.

У Godot в корне находится `project.godot`, который сообщает движку, что перед ним проект и где искать основные настройки. Рядом могут лежать `main.tscn`, `player.tscn`, `player.gd`, папки `scenes`, `scripts`, `assets`. У Unity особенно важны `Assets`, `Packages` и `ProjectSettings`, а пользовательские скрипты обычно находятся где-нибудь внутри `Assets`. В GDevelop проект тоже хранит конфигурацию и ссылки на ресурсы, хотя значительная часть логики редактируется визуально.

Полезно иногда открывать папку в обычном файловом

менеджере. После того как редактор создал новую сцену или скрипт, найдите соответствующий файл на диске. Это простое действие быстро разрушает ощущение магии. Вы начинаете видеть, что программа состоит из вполне обычных элементов, а агент всего лишь меняет их быстрее человека.

Глава 9. Как ставить задачу, чтобы не переделывать весь проект

Большая часть неудачного вайбкодинга начинается с слишком большого запроса. Просьба создать RPG с открытым миром, прокачкой, крафтом, торговлей, диалогами и умными NPC технически понятна, но скрывает сотни решений, которые модель вынуждена принять самостоятельно. Даже если результат запускается, он почти наверняка будет устроен не так, как представлял пользователь.

Полезнее описывать один завершённый игровой цикл или пользовательский сценарий. Для аркады достаточно персонажа, одного вида опасности, условия победы или поражения и перезапуска. Для приложения заметок достаточно создания, редактирования, списка и сохранения. Когда этот слой проверен, можно добавлять следующий.

Хороший запрос описывает наблюдаемое поведение. Вместо «сделай нормальную архитектуру движения» полезнее написать: «Персонаж двигается WASD, скорость по диагонали должна совпадать с обычной, после отпущения клавиши он останавливается, а за границу уровня выйти нельзя». Вы не обязаны знать, какая функция или математическая операция нужна для реализации, но обязаны понимать, что именно должно происходить на экране.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.